1/4-Inch 3.1Mp System-On-A-Chip (SOC) CMOS Digital Image Sensor

MT9T111

For the latest data sheet, refer to Aptina's Web site: www.aptina.com

About This Document

This developer guide is a reference for hardware and software engineers developing camera systems using the Aptina® MT9T111 CMOS digital image sensor. The MT9T111 is a complete system-on-a-chip (SOC) image sensor that integrates seamlessly in today's mobile phone applications. It incorporates sophisticated on-chip camera functions and is programmable through a serial interface. This document provides information on hardware interfaces, camera control, and register programming recommendations to optimize image quality.

The system configuration section provides system-level information about the MT9T111. This information is intended for module integrators or board-level design engineers. It covers the MT9T111's signal descriptions, system connections, power supply configuration, and I/O signal states.

The architecture overview describes functions of each major block and their related register descriptions inside the MT9T111. It covers the PLL, the two-wire serial interface, the output interface, GPIO signals, and the OTP memory.

The programming and operation section provides the programming procedure and initialization process of the MT9T111. It covers how to access internal registers and variables using the two-wire serial interface. It also covers the power-on initialization process, PLL programming, and standby mode operation. Example code is included.

The image signal processing flow and camera control section describes a variety of features of the MT9MT111 including: auto focus (AF), anti-shaking, auto exposure (AE), auto white balance (AWB), flicker detection (FD), JPEG output, lens shading correction (LC), gamma correction, and context switching. Related registers are included.

The development tool overview provides steps to calibrate the MT9T111 for timing, lens shading, and color tuning. It also covers how to use the Register Wizard and DevWare tools.

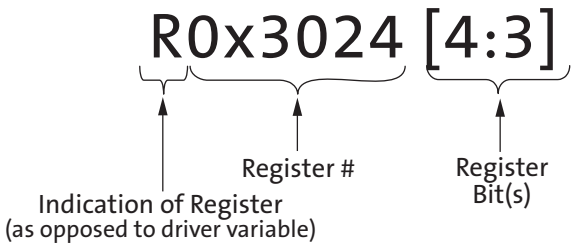
There are two appendices that provide application examples for using the MT9T111 in a dual camera system and in the demo board system environment. Appendix A describes the system-level connections with a secondary sensor for dual camera mode. Appendix B provides a high-level board-to-board interconnection description of the demo board system.

Note: This developer guide is applicable to the MT9T111's Rev3 silicon.

Introduction to Registers

This developer guide refers to various memory locations and registers that the user reads from or writes to for altering the MT9T111 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 utilizing hardware registers from 0x098E through 0x0990 (see the MT9T111 data sheet for further details on how to use these registers). These are denoted below:

Figure 2: Firmware Variable Legend

The MT9T111 was designed to facilitate customizations to optimize image quality processing. As the image data travels 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.

Accessing the Firmware Drivers' Variables

R0x098E is used for the memory address and R0x0990 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, the data is available when read from the indirect access data register.

Refresh Mode and Refresh Commands

When some register values are changed, for example registers for cropping and zooming, a refresh mode command (seq.seq_cmd = 6) and a refresh command (seq.seq_cmd = 5) must be issued before the new settings will take effect. Aptina recommends that the commands be issued sequentially (refresh mode then refresh). Refer to the data sheet to determine which other registers need this command sequence.

MCU Memory

R0x098E[15:0], Indirect Access Address Register
R0x0990[15:0], Indirect Access Data Register

Table 1: R0x098E[15:0], Indirect Access Address Register

Bit	Description
15	logical_word_access This bit specifies the type of access. 0: Word access (2 bytes) 1: Byte access
14:10	This field specifies the driver number for the logical access.
9:0	This field specifies the offset for the logical access.

Conventions and Notations

This developer guide follows the conventions and notations described below.

- Hexadecimal numbers have 0x prefix
- Binary numbers have 0b prefix
  Example: 0b1010 = 0xA
- Fixed point notation
  0.8 (0.0 through 254/255)
  1.7 (0.0 through 1 + 127/128)
- I/O signals can be LOW (0 or DGND), HIGH (1 or VDD_IO), or floating (high impedance or High-Z)
- Timing diagrams are not drawn to scale and do not illustrate the actual number of clocks necessary.

Table of Contents

List of Figures

List of Tables

Overview

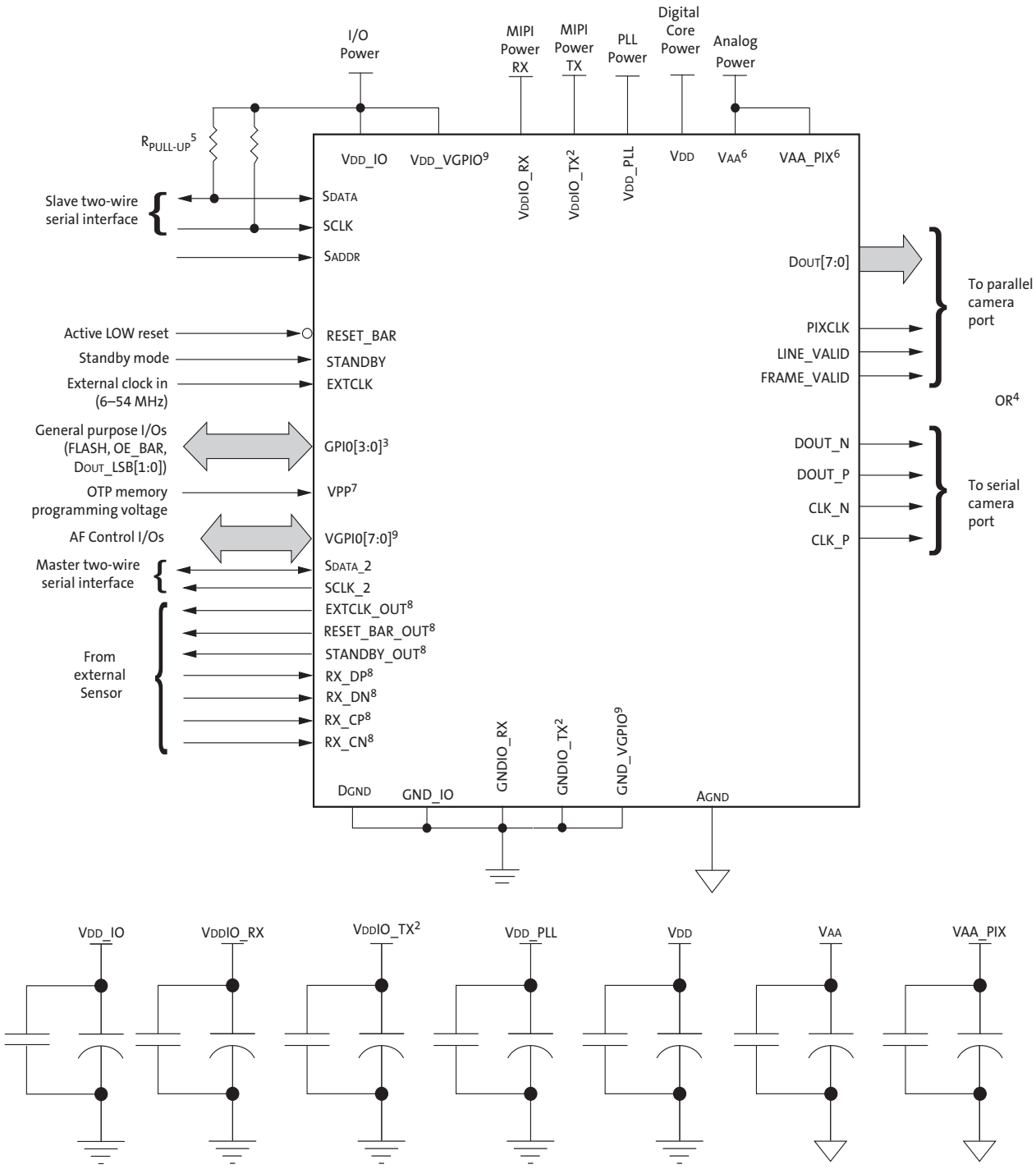
The MT9T111 is a 3.1Mp CMOS sensor with Aptina's the latest image signal processing (ISP) and 1.75µm pixel technology. The MT9T111's features include:

- 2048 x 1536 visible pixels in 1.75µm technology
- 30 fps maximum with 1024 x 768 YUV output
- 15 fps maximum with 2048 x 1536 JPEG output
- Parallel output, MIPI transmitter, MIPI receiver, and two-wire serial interface
- Dual camera support with the MT9V013 or the MT9V113
- Anti-Shake and auto focus features
- JPEG and thumbnail through parallel and MIPI output
- Fully enhanced next-generation ISP engine
- Hard standby, with and without memory retention
- Soft standby with memory retention

System Configuration

Figure 3 shows typical device connections for the MT9T111.

Figure 3: Typical Configuration (connection)



Notes:
1. This typical configuration shows only one scenario (out of multiple possible variations) for this sensor.
2. If a MIPI Interface is not required, the VDDIO_TX and VDDIO_RX pads must be connected to VDD and the GNDIO_TX and GNDIO_RX pads must be connected to DGND. The following signals must be left floating: DOUT_P, DOUT_N, CLK_P, and CLK_N.
3. The GPIO pads can serve multiple features that can be reconfigured. The function and direction will vary by applications.
4. Only one of the output modes (serial or parallel) can be used at any time.
5. Aptina recommends a resistor value of 1.5KΩ to VDD_IO for the two-wire serial interface RPULL-UP; however, greater values may be used for slower transmission speed.
6. VAA and VAA_PIX must be tied together.
7. VPP is the one-time programmable (OTP) memory programming voltage and should be left floating during normal operation.
8. If the bridging function to the MT9V013 is not required, the following signals can be left floating: EXTCLK_OUT, RESET_BAR_OUT, STANDBY_OUT, RX_DP, RX_DN, RX_CP, and RX_CN.
9. If auto focus (AF) is not required, the following pads can be left floating: VDD_VGPIO, GND_VGPIO, and VGPIO[7:0].
10. Aptina recommends that 0.1µF and 1µF decoupling capacitors for each power supply are mounted as close as possible to the pad. Actual values and results may vary, depending on layout and design considerations.

Signal Description

Table 2 provides the signal descriptions for the MT9T111.

Table 2: Signal Descriptions

Name	Type	Description
STANDBY	Input	Controls sensor's standby mode, active HIGH.
RESET_BAR	Input	Master reset signal, active LOW (can be left floating if not used).
EXTCLK	Input	Input clock signal 6–54 MHz.
RX_DN	Input	Differential MIPI data from the MT9V013 (sub-LVDS, negative) (must be left floating if not used).
RX_DP	Input	Differential MIPI data from the MT9V013 (sub-LVDS, positive) (must be left floating if not used).
RX_CN	Input	Differential MIPI clock from the MT9V013 (sub-LVDS, negative) (must be left floating if not used).
RX_CP	Input	Differential MIPI clock from the MT9V013 (sub-LVDS, positive) (must be left floating if not used).
VPP	Input	High voltage programming signal for anti-fuse of module ID and other features (must be left floating for normal operation).
SCLK	Input	Slave two-wire serial interface clock from the host processor.
SADDR	Input	Selects device address for the slave two-wire serial interface. The address is 0x78 when SADDR is tied LOW, 0x7A if tied HIGH.
SDATA	I/O	Slave two-wire serial interface data to and from the host processor.
SCLK_2	Output	Master two-wire serial interface clock to the MT9V013 and peripheral devices, such as AF mechanics.
SDATA_2	I/O	Master two-wire serial interface data to and from the MT9V013 and peripheral devices, such as AF mechanics.
GPIO[3:0]	I/O	General purpose digital I/O, could be configured for FLASH/SHUTTER/DOUT_LSB0/DOUT_LSB1/ OE_BAR.
VGPIO[7:0]	I/O	General purpose digital I/O, used for AF function (can be left floating if not used).
DOUT[7:0]	Output	Eight-bit image data output or most significant bits (MSB) of 10-bit SOC bypass mode.
PIXCLK	Output	Pixel clock. Used for sampling DOUT, FRAME_VALID, and LINE_VALID.
LINE_VALID	Output	Identifies pixels in the active line.
FRAME_VALID	Output	Identifies rows in the active image.
DOUT_N	Output	Differential MIPI data (sub-LVDS, negative) (must be left floating if not used).
DOUT_P	Output	Differential MIPI data (sub-LVDS, positive) (must be left floating if not used).
CLK_N	Output	Differential MIPI clock (sub-LVDS, negative) (must be left floating if not used).
CLK_P	Output	Differential MIPI clock (sub-LVDS, positive) (must be left floating if not used).
RESET_BAR_OUT	Output	Master reset signal to the MT9V013, active LOW.
EXTCLK_OUT	Output	Input clock signal to the MT9V013.
STANDBY_OUT	Output	Standby mode control for the MT9V013, active HIGH.
VDD	Supply	Digital power (1.8V typical).
VAA_PIX	Supply	Pixel array power (2.8V typical).
VAA	Supply	Analog power (2.8V typical).
VDD_PLL	Supply	PLL power (2.8V typical).
VDD_IO	Supply	I/O power supply (1.8V or 2.8V typical).
GND_IO	Supply	I/O ground.
DGND	Supply	Digital ground.
AGND	Supply	Analog ground.
VDDIO_TX	Supply	I/O power supply for the MIPI output interface, 1.8V typical (must be connected to VDD, even if the interface is not used).
GNDIO_TX	Supply	I/O ground supply for the MIPI output interface (must be connected to DGND even if the interface is not used).

Table 2: Signal Descriptions (continued)

Name	Type	Description
VDDIO_RX	Supply	I/O power supply for the MIPI input interface, 1.8V typical (must be connected to VDD, even if the interface is not used).
GNDIO_RX	Supply	I/O ground supply for the MIPI input interface (must be connected to DGND, even if the interface is not used).
VDD_VGPIO	Supply	I/O power supply for VGPIO[7:0] signals (can be left floating if the interface is not used).
GND_VGPIO	Supply	I/O ground for VGPIO[7:0].

I/O Signals

The MT9T111 has digital I/O signals like a typical CMOS circuit.

- I/O voltage (VDD_IO) is separate from core voltages (VDD and VDD_PLL) and analog voltages (VAA and VAA_PIX).
- If an input signal is not used, it must be tied to HIGH (VDD_IO) or LOW (DGND).
- Input signals with internal pull-up or pull-down resistors can be left floating (see Table 3 for details).
- If an output signal is not used, leave it floating.
- If a bidirectional signal is not used, it must be either configured as an output, or if configured as an input, it must be tied to either HIGH or LOW.
- Never exceed the absolute maximum electrical specification (refer to the electrical specifications section of the MT9T111 data sheet).

In general, when VDD_IO is on, all the input signals must be driven by the host, while the output signals are driven by the sensor. The MT9T111 implements a fail-safe I/O for all I/O signals to prevent I/O failure when VDD_IO is not applied.

If some of the features are not used in the sensor, the input signals must be connected to either HIGH or LOW, as shown in Table 3.

Table 3: Attributes of I/O and Power Supply Signals

Signal Name	Type	Supply Reference	Fail–Safe I/O	If Unused	Slew Rate	Hysteresis Input	Internal Pull-Up/ Down
EXTCLK	Input	VDD_IO/DGND	Yes	N/A	N/A	Yes	No
RESET_BAR	Input	VDD_IO/DGND	Yes	N/A	N/A	N/A	Yes (PU)
SADDR	Input	VDD_IO/DGND	Yes	N/A	N/A	N/A	No
SCLK	Input	VDD_IO/DGND	Yes	N/A	N/A	Yes	No
STANDBY	Input	VDD_IO/DGND	Yes	DGND	N/A	N/A	No
RX_DN	Input	VDDIO_RX/ GNDIO_RX	N/A	No connection	N/A	N/A	No
RX_DP	Input	VDDIO_RX/ GNDIO_RX	N/A	No connection	N/A	N/A	No
RX_CN	Input	VDDIO_RX/ GNDIO_RX	N/A	No connection	N/A	N/A	No
RX_CP	Input	VDDIO_RX/ GNDIO_RX	N/A	No connection	N/A	N/A	No
GPIO[3:0]	I/O	VDD_IO/DGND	Yes	DGND or VDD_IO	R0x001E[6:4]	N/A	No
SDATA[1]	I/O	VDD_IO/DGND	Yes	N/A	N/A	Yes	No
SDATA_2[1]	I/O	VDD_IO/DGND	N/A	No connection	N/A	N/A	No
VGPIO[7:0]	I/O	VDD_VGPIO/ GND_VGPIO	N/A	No connection	N/A	N/A	No
DOUT[7:0]	Output	VDD_IO/DGND	Yes	No connection	R0x001E[2:0]	N/A	No
FRAME_VALID	Output	VDD_IO/DGND	Yes	No connection	R0x001E[2:0]	N/A	No
LINE_VALID	Output	VDD_IO/DGND	Yes	No connection	R0x001E[2:0]	N/A	No
PIXCLK	Output	VDD_IO/DGND	Yes	No connection	R0x001E[10:8]	N/A	No
DOUT_P	Output	VDDIO_TX/ GNDIO_TX	N/A	No connection	N/A	N/A	No
DOUT_N	Output	VDDIO_TX/ GNDIO_TX	N/A	No connection	N/A	N/A	No

Table 3: Attributes of I/O and Power Supply Signals (continued)

Signal Name	Type	Supply Reference	Fail–Safe I/O	If Unused	Slew Rate	Hysteresis Input	Internal Pull-Up/ Down
CLK_P	Output	VDDIO_TX/ GNDIO_TX	N/A	No connection	N/A	N/A	No
CLK_N	Output	VDDIO_TX/ GNDIO_TX	N/A	No connection	N/A	N/A	No
SCLK_2	Output	VDD_IO/DGND	N/A	No connection	N/A	N/A	No
RESET_BAR_OUT	Output	VDD_IO/DGND	N/A	No connection	N/A	N/A	No
EXTCLK_OUT	Output	VDD_IO/DGND	N/A	No connection	N/A	N/A	No
STANDBY_OUT	Output	VDD_IO/DGND	N/A	No connection	N/A	N/A	No
AGND	Supply	VAA, VAA_PIX	N/A	N/A	N/A	N/A	No
DGND	Supply	VDD_IO/DGND	N/A	N/A	N/A	N/A	No
VAA	Supply	AGND	N/A	N/A	N/A	N/A	No
VAA_PIX	Supply	AGND	N/A	N/A	N/A	N/A	No
VDD	Supply	DGND	N/A	N/A	N/A	N/A	No
VDD_IO	Supply	DGND	N/A	N/A	N/A	N/A	No
VDD_PLL	Supply	DGND	N/A	N/A	N/A	N/A	No
VPP	Supply	DGND	N/A	No connection	N/A	N/A	No
VDDIO_TX	Supply	GNDIO_TX	N/A	VDD	N/A	N/A	No
GNDIO_TX	Supply	VDDIO_TX	N/A	DGND	N/A	N/A	No
VDDIO_RX	Supply	GNDIO_RX	N/A	VDD	N/A	N/A	No
GNDIO_RX	Supply	VDDIO_RX	N/A	DGND	N/A	N/A	No
VDD_VGPIO	Supply	GND_VGPIO	N/A	No connection	N/A	N/A	No
GND_VGPIO	Supply	VDD_VGPIO	N/A	No connection	N/A	N/A	No

Notes: 1. Output is open drain.

Table 4 shows the MT9T111 output states during reset and standby.

Table 4: Output Signal States During Reset and Standby

Signal	Reset	Standby
DOUT[7:0]	High-Z	High-Z
PIXCLK	High-Z	High-Z
LINE_VALID	High-Z	High-Z
FRAME_VALID	High-Z	High-Z
DATA_N	0	0
DATA_P	0	0
CLK_N	0	0
CLK_P	0	0
GPIO[3:0]	High-Z	High-Z
VGPIO[7:0]	High-Z	High-Z
RESET_BAR_OUT	High-Z	High-Z
EXTCLK_OUT	High-Z	High-Z
STANDBY_OUT	High-Z	High-Z

Low-Noise Operation

For low-noise operation, the MT9T111 requires separate power supplies for analog and digital. Incoming digital and analog ground conductors need to be separated in the imager module. Both power supply rails should be decoupled from ground, using capacitors located as close as possible to the die. The use of inductance filters is not recommended on the power supplies or output signals.

The MT9T111 also supports different digital core, MIPI, and I/O power domains, as shown in Table 5. The PLL and analog domains require clean power sources.

Table 5: Power Supply Descriptions

Voltage Name	Description	Voltage	Ground Reference	Note
VDD	Digital logic	1.8V	DGND	1, 2
VDD_IO	Input/output	1.8V or 2.8V	GND_IO	1, 2
VPP	OTP memory	8.5V	DGND	–
VDD_PLL	PLL	2.8V	DGND	2
VDDIO_Tx	MIPI TX	1.8V	GNDIO_TX	–
VDDIO_RX	MIPI RX	1.8V	GNDIO_RX	–
VDDIO_VGPIO	VGPIO	2.8V	GNDIO_VPGIO	–
VAA	Analog circuitry	2.8V	AGND	2
VAA_PIX	Pixel array	2.8V	AGND	2

Notes: 1. DGND and GND_IO are tied together to meet ESD standards.
2. AGND and DGND are all isolated internally.

If there are five pins available for system power connections, Aptina recommends the connections, as shown in Table 6.

Table 6: Recommended System Power Connections (5 pins)

Pin Number	Power/Ground Type	Voltage	Power Supply Signals
1	Digital power	1.8V	VDD, VDDIO_TX, VDDIO_RX
2	Digital power	1.8V or 2.8V	VDD_IO, VDDIO_VGPIO
3	Analog power	2.8V	VAA, VAA_PIX, VDD_PLL
4	DGND		DGND, GND_IO, GNDIO_TX, GNDIO_RX, GNDIO_VGPIO
5	AGND		AGND

If there are only four pins available for system power connections, Aptina recommends the connections, as shown in Table 7.

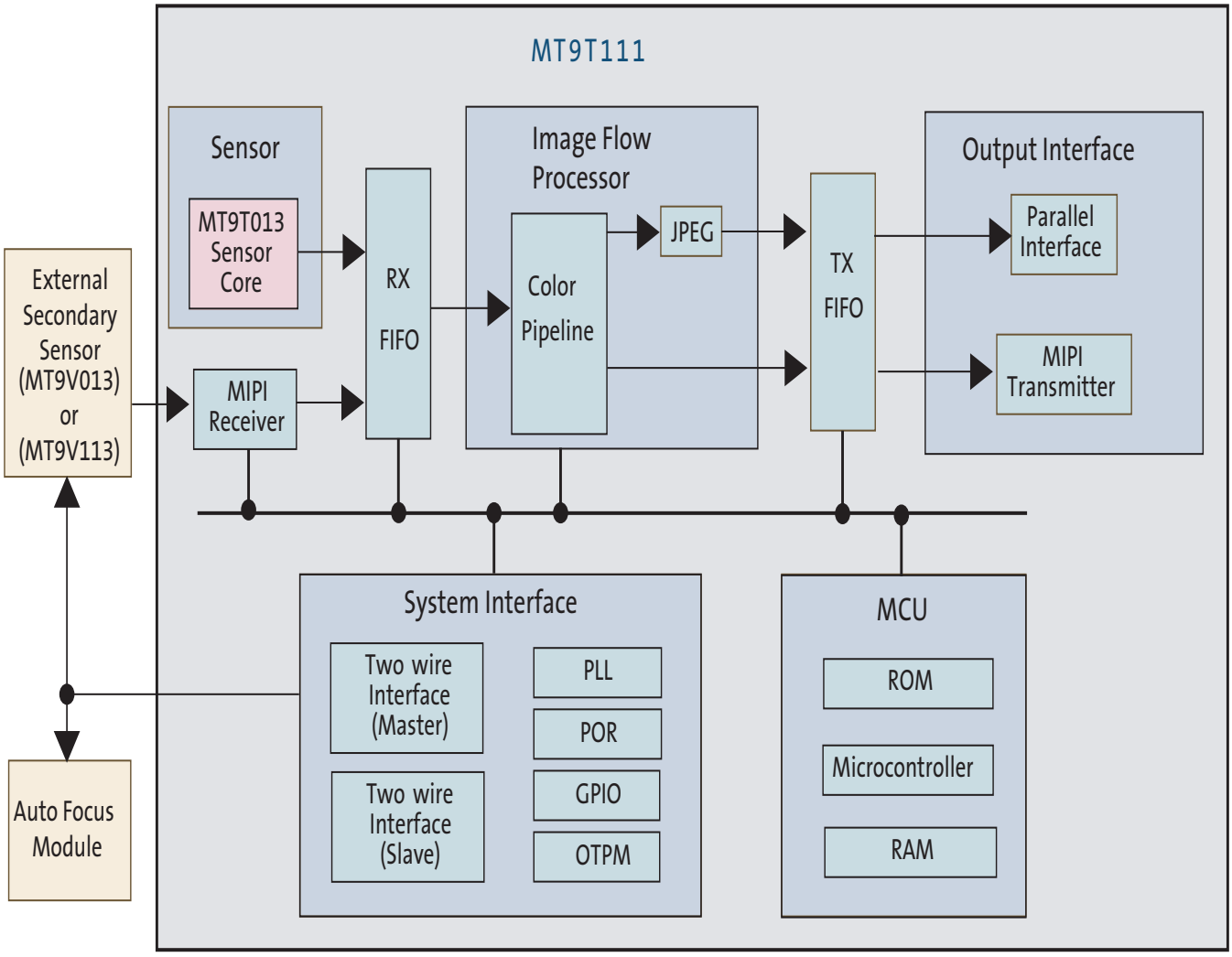
Table 7: Recommended System Power Connections (4 pins)

Pin Number	Power Type	Voltage	Power Supply Signals
1	Digital power	1.8V	VDD, VDD_IO, VDDIO_TX, VDDIO_RX, VDDIO_VGPIO
2	Analog power	2.8V	VAA, VAA_PIX, VDD_PLL
3	DGND		DGND, GND_IO, GNDIO_TX, GNDIO_RX, GNDIO_VGPIO
4	AGND		AGND

Architecture Overview

The MT9T111 combines a 3.1Mp sensor core with an image processor (image flow pipeline) to form a stand-alone solution that includes both image acquisition and processing, as shown in Figure 4. Both the sensor core and the image flow pipeline have internal registers that can be controlled by the user. In normal operation, an integrated microcontroller autonomously controls most aspects of operation. The processed image data is transmitted to the host system, either through a parallel bus or a serial data bus through the output interface.

Figure 4: SOC Block Diagram



PLL and Clock Divider

The MT9T111 requires a single input clock source (EXTCLK) to operate. EXTCLK is used as the reference clock to the PLL, which generates all the internal clocks. The sensor provides three output clocks (PIXCLK, external sensor CLK, and MIPI CLK). PIXCLK is used for the parallel output data bus, whereas the MIPICLK is used for the serial MIPI output. The MT9T111 also provides the clock for an external sensor to be used in dual camera applications.

The PLL can accept an input clock with a frequency between 6 and 54 MHz. A PLL circuit in the MT9T111 synthesizes an arbitrary clock frequency to be used as the time base within the device. It allows flexible system design where the clock supplied to the device through EXTCLK may be higher or lower than the desired PIXCLK output frequency.

On power-up, the PLL is bypassed and the MT9T111 initially runs off of the EXTCLK. When the PLL is enabled, the MT9T111 uses the PLL output (VCO) clock to generate all the internal clocks. All of the clocks shown in Figure 5 on page 22 are derived from the VCO (see Table 10 on page 23). Table 9 on page 22 shows how to calculate the internal clock frequencies.

The PLL divider registers must be programmed before the PLL is enabled. Aptina recommends using the MT9T111's Register Wizard (part of DevWare), a frame rate and timing calculating tool, to generate the register values. The user only needs to specify input clock frequency (EXTCLK), desired output clock frequency (PIXCLK), and frame rate requirements.

The sensor clock will be calculated based on frame rate requirements. Based on sensor clock frequency and the required pixel clock, the SOC clock frequency is calculated. The SOC clock frequency should be as low as possible to provide the lowest power consumption, but still maintain continuous data flow from sensor to the output interface. A RX FIFO and a TX FIFO with programmable watermarks allows for the continuous flow of image data.

For programming the external sensor clock, refer to "Master Two-Wire Serial Interface and External Sensor Control Interface" on page 37.

For more detailed programming of the sensor core clock with frame rate calculation, refer to the MT9T013 1/4-Inch 3.1Mp CMOS digital image sensor data sheet.

PIXCLK and DOUT[7:0] Timing

By default in Rev3, pixel data (DOUT[7:0]) is output on the falling edge of PIXCLK. PIXCLK in default mode is commonly referred to as an "inverted" PIXCLK because data changes on the falling edge. The host controller should capture data on the rising edge of PIXCLK while LV = 1. The PIXCLK polarity may be reversed by programming R0x3C20 in the TX_SS register as shown in Table 8.

Table 8: Polarity of PIXCLK

R0x3C20[0] Value	Sensor Outputs Data	Host Captures Data
0	Rising edge	Falling edge
1 (default Rev3)	Falling edge	Rising edge

Figure 5: Clock Distribution

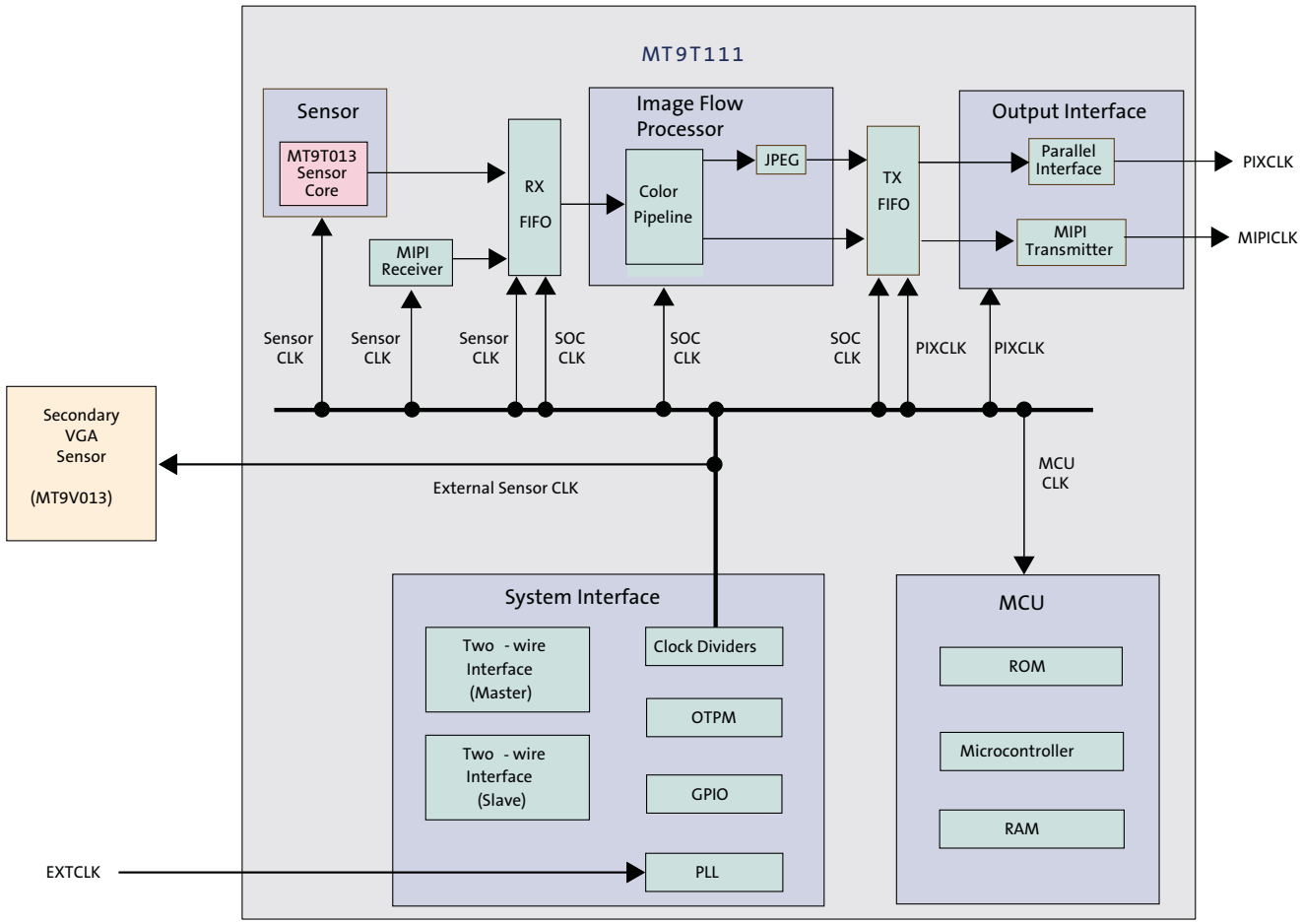


Table 9: MT9T111 Clock Calculation Summary

Name	Description	Min Freq (MHz)	Max Freq (MHz)	Equation
EXTCLK	System input clock to MT9T111	6	54	
PIXCLK	Output pixel clock	–	96	VCO / [(P1 + 1) * (P2 + 1)]
MIPICLK	Output MIPI clock	–	768	VCO / (P3 + 1)
MCU CLK	Internal clock for MCU block	–	96	VCO / (P6 + 1)
SOC CLK	Internal clock for color pipe block	–	54	VCO / (P5 + 1)
Sensor CLK	Internal clock for sensor core clock	–	70	VCO / (P4 + 1)
External sensor CLK	Output clock for external sensor	–	–	VCO / (P7 + 1)
VCO output	PLL internal clock	384	768	2 * M * EXTCLK / (N + 1)
f_{PFD}	PLL internal clock	2	24	EXTCLK / (N + 1)

Table 10 shows the registers related to programming the PLL and clock dividers. Refer to the MT9T111 data sheet for more detailed descriptions of registers and variables.

Table 10: PLL and Clock Related Registers and Variables

Map	Address	Bits	Descriptions
SYSCTL	0x0010	[13:8]	N divider value for VCO.
SYSCTL	0x0010	[7:0]	M multiplier value for VCO.
SYSCTL	0x0012	[11:8]	P3 divider for MIPI CLK.
SYSCTL	0x0012	[7:4]	P2 divider for PIXCLK.
SYSCTL	0x0012	[3:0]	P1 divider for PIXCLK.
SYSCTL	0x0014	[15]	PLL lock status. Read-only. This bit indicates when the PLL has acquired lock. After the PLL is enabled, and the internal counters are reset, this bit goes HIGH when PLL lock is detected. 0: PLL lock is not detected 1: PLL lock is detected
SYSCTL	0x0014	[8]	Internal PLL counter reset. Before PLL is locked, set to "0." After PLL is locked, reset the counters so the PLL will achieve lock before it is used. This bit should be set to "1" after PLL is locked.
SYSCTL	0x0014	[1]	PLL enable bit. Active HIGH. 0: Disable PLL 1: Enable PLL
SYSCTL	0x0014	[0]	PLL bypass. Set to start the initialization of PLL. After PLL is locked, reset to "0" for normal operation. 0: PLL bypass disable 1: PLL bypass enable
SYSCTL	0x0016	[9]	Enable EXTCLK input port. 0: Disable 1: Enable for normal operation
SYSCTL	0x002A	[11:8]	P6 divider for MCU CLK.
SYSCTL	0x002A	[7:4]	P5 divider for SOC CLK.
SYSCTL	0x002A	[3:0]	P4 divider for sensor CLK.
SYSCTL	0x002C	[3:0]	P7 divider for external sensor CLK.

PLL Bypass Mode

The MT9T111 provides the option to use EXTCLK as an internal clock source instead of the internal PLL clock. During PLL bypass mode, EXTCLK source will bypass the internal PLL block and P divider sections and drive all the internal clocks. EXTCLK drives all the clocks shown in Figure 5 on page 22. After power-up, SYSCTL 0x0014[0] is set to "1" to indicate bypass mode. The user can skip the PLL programming procedure and leave SYSCTL 0x0014[0] set to "1."

Low Power Mode

The MT9T111 supports low power operation during preview mode by reducing the pixel clock frequency. When the MT9T111 enters preview mode with low power mode enabled (Cam_Pri Context A variable 0x000C[9] = 1), the sensor clock will be reduced by half. Internal logic will disable the pixel clock and re-program the divider value. Internal delay will be applied during this change to avoid any clock glitches.

Example of Code to Program PLL and Clocks

The following .ini code example shows how to program the PLL and initialize the clocks:

```
[PLL Setup - 16 MHz External]

//  (From Wizard)
//; Input Frequency: 16.000 MHz
//; VCO Frequency:  768.000 MHz
//; P2 = 8
//; P4 = 11
//; P5 = 15
//; P6 = 8
//; Pads Clock Frequency:   VCO/P2 = 96.000 MHz
//; Sensor Core Clock Freq: VCO/P4 = 69.818 MHz
//; SOC Clock Frequency:    VCO/P5 = 51.200 MHz
//; MCU Clock Frequency:    VCO/P6 = 96.000 MHz
BITFIELD = 0x14, 1, 1       // Bypass PLL
BITFIELD = 0X14, 2, 0       // Power-down PLL
REG = 0x0014, 0x2145        // PLL control: BYPASS PLL
REG = 0x0010, 0x0018        // PLL Dividers
REG = 0x0012, 0x0070        // PLL P Dividers 1-2-3
REG = 0x002A, 0x77EA        // PLL P Dividers 4-5-6
REG = 0x0014, 0x2545        // PLL control: TEST_BYPASS on
REG = 0x0014, 0x2547        // PLL control: PLL_ENABLE on
REG = 0x0014, 0x3447        // PLL control: SEL_LOCK_DET
REG = 0x0014, 0x3047        // PLL control: TEST_BYPASS off

POLL_REG=0x0014,0x8000 == 0x0000, DELAY=10, TIMEOUT = 5000

REG = 0x0014, 0x3046        // PLL control: PLL_BYPASS off
REG = 0x0022, 320       // reference clock count for 20 us
REG = 0x0016, 0x0400
```

[PLL Setup - 24MHz External]

```
//  (From Wizard)
//; Input Frequency: 24.000 MHz
//; VCO Frequency:  768.000 MHz
//; P2 = 8
//; P4 = 11
//; P5 = 15
//; P6 = 8
//; Pads Clock Frequency:   VCO/P2 = 96.000 MHz
//; Sensor Core Clock Freq: VCO/P4 = 69.818 MHz
//; SOC Clock Frequency:    VCO/P5 = 51.200 MHz
//; MCU Clock Frequency:    VCO/P6 = 96.000 MHz
BITFIELD = 0x14, 1, 1       // Bypass PLL
BITFIELD = 0X14, 2, 0       // Power-down PLL
REG = 0x0014, 0x2145        // PLL control: BYPASS PLL
REG = 0x0010, 0x0010        // PLL Dividers
REG = 0x0012, 0x0070        // PLL P Dividers 1-2-3
REG = 0x002A, 0x77EA        // PLL P Dividers 4-5-6
REG = 0x0014, 0x2545        // PLL control: TEST_BYPASS on
REG = 0x0014, 0x2547        // PLL control: PLL_ENABLE on
REG = 0x0014, 0x3447        // PLL control: SEL_LOCK_DET on
REG = 0x0014, 0x3047        // PLL control: TEST_BYPASS off

POLL_REG=0x0014,0x8000 == 0x0000, DELAY=10, TIMEOUT = 5000

REG = 0x0014, 0x3046        // PLL control: PLL_BYPASS off
REG = 0x0022, 480       // reference clock count for 20 µs
```

RX and TX FIFO Watermark

The MT9T111 has two internal FIFO blocks to support different clock domains. The RX FIFO handles the data flow from the sensor to the IFP, and the TX FIFO handles the data flow from the IFP to the output interface block, as shown in Figure 6.

The watermark indicates the level of the FIFO to start the read out operation for continuous image data flow. Both FIFOs must be programmed with correct watermark data to operate correctly, as shown in Table 11. The Register Wizard tool generates the proper watermark values for both RX and TX FIFOs for different clock settings and output image sizes. The MT9T111 programs the TX FIFO watermark automatically, but also allows the user to program the watermark values manually. To activate the manual override, the user must program the TX FIFO watermark and set the manual flag, as shown in Table 11. The MT9T111 does not calculate the RX FIFO watermark, and always uses the user programmed value.

Figure 6: Watermark Block Diagram

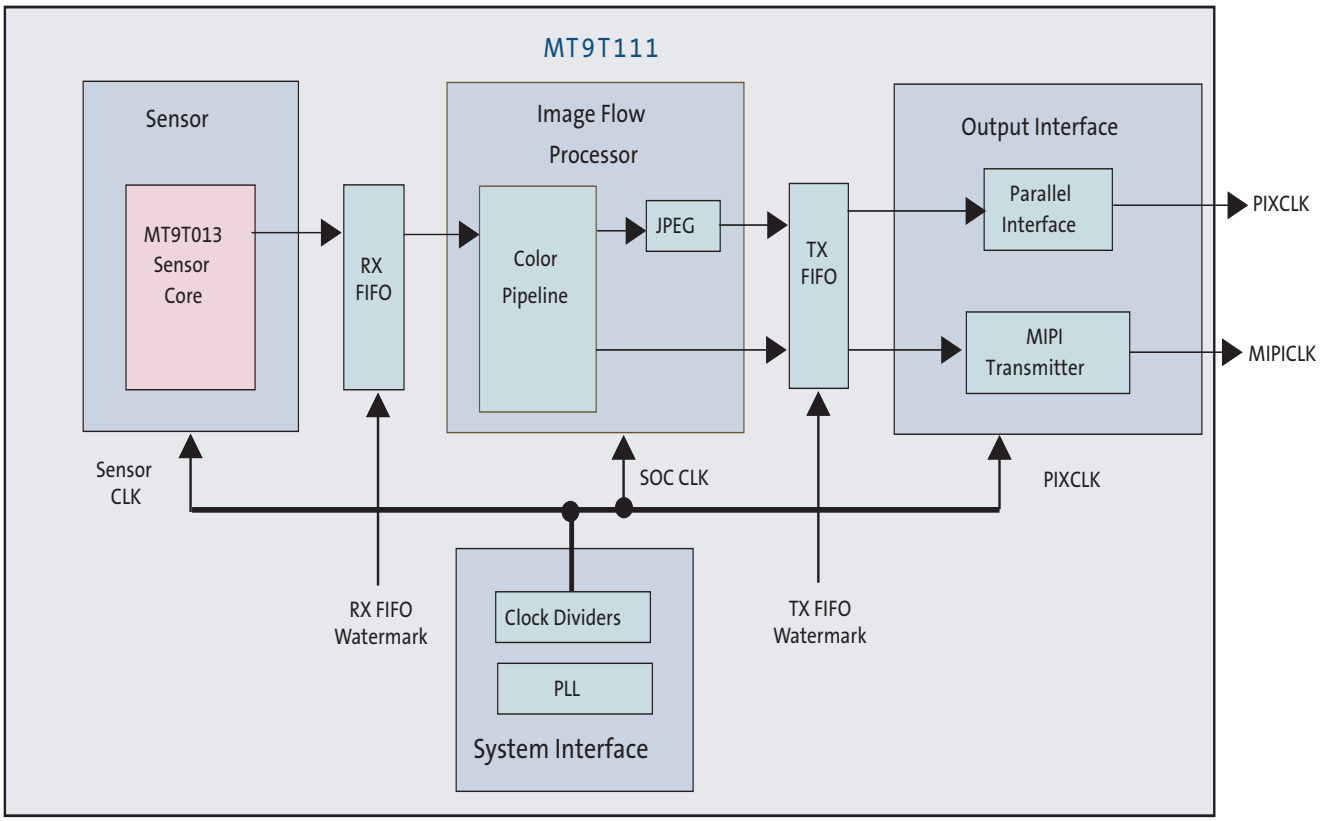


Table 11: Watermark Values to be Programmed

Map	Address for 0x098E	Bits	Description
CAM1 Control	0x4846	[15:0]	Camera1 (internal sensor) context A RX FIFO watermark value.
CAM1 Control	0x4882	[15:0]	Camera1 (internal sensor) context B RX FIFO watermark value.
CAM2 Control	0x4C46	[15:0]	Camera2 (external sensor) context A RX FIFO watermark value.

Table 11: Watermark Values to be Programmed (continued)

Map	Address for 0x098E	Bits	Description
CAM2 Control	0x4C82	[15:0]	Camera2 (external sensor) context B RX FIFO watermark value.
Cam_Pri Context A	0x68AA	[15:0]	Camera1 (internal sensor) context A TX FIFO manual watermark value.
Cam_Pri Context A	0xE8AC	[0]	Camera1 (internal sensor) context A TX FIFO manual flag. 0: Disable 1: Enable
Cam_Pri Context B	0x6CAA	[15:0]	Camera1 (internal sensor) context B TX FIFO manual watermark value.
Cam_Pri Context B	0xECAC	[0]	Camera1 (internal sensor) context B TX FIFO manual flag. 0: Disable 1: Enable
Cam_Sec Context A	0x70AA	[15:0]	Camera2 (external sensor) context A TX FIFO manual watermark value.
Cam_Sec Context A	0xF0AC	[0]	Camera2 (external sensor) context A TX FIFO manual flag. 0: Disable 1: Enable
Cam_Sec Context B	0x74AA	[15:0]	Camera2 (external sensor) context B TX FIFO manual watermark value.
Cam_Sec Context B	0xF4AC	[0]	Camera2 (external sensor) context B TX FIFO manual flag. 0: Disable 1: Enable

Example of .ini File to Program the Watermark

```
REG = 0x98E, 0x4846 //  RX FIFO Watermark (A)
REG = 0x990, 0x0014 // = 20
REG = 0x98E, 0x68AA //  TX FIFO Watermark (A)
REG = 0x990, 0x0510 // = 1296
REG = 0x98E, 0xE8AC //  TX FIFO Manual Watermark (A)
REG = 0x990, 0x01 // = 1
REG = 0x98E, 0x488E //  RX FIFO Watermark (B)
REG = 0x990, 0x0014 // = 20
REG = 0x98E, 0x6CAA //  TX FIFO Watermark (B)
REG = 0x990, 0x008A // = 138
REG = 0x98E, 0xECAC //  TX FIFO Manual Watermark (B)
REG = 0x990, 0x01 // = 1
REG = 0x98E, 0x8400 //  Refresh Sequencer Mode
REG = 0x990, 0x06 // = 6
```

Output Slew Rate Control

System power consumption, noise, and electromagnetic interference (EMI) are reduced by selecting the optimum slew rate to meet the timing budget.

The MT9T111 output slew rate control register is discussed in the SYSCTL register section of the MT9T111 data sheet. R0x001E[2:0] controls the DOUT[7:0], FRAME_VALID (FV), LINE_VALID (LV) slew rate. Bits [6:4] control the GPIO[3:0] output slew rate. Bits [10:8] control the slew rate of PIXCLK, independent of other outputs. Eight slew rates (code 0–code 7) are selectable. Code 0 is the slowest; code 7 is the fastest. Rise time and fall time are matched.

SDATA and SCLK have no rise time slew rate control. SDATA has an open drain output without an active p-channel transistor. Slew rate control is accomplished by an external passive resistor.

Figure 7 shows how slew rate is measured.

Figure 7: Output Slew Rate Defined

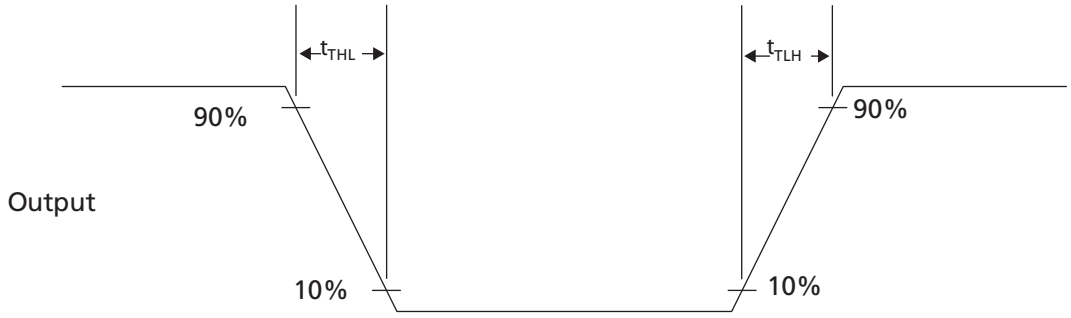


Table 12 shows the related registers to program slew rate for output signals. Refer to the MT9T111 data sheet for more detailed AC/DC electrical specifications.

Table 12: Slew Rate Control Related Registers

Map	Address	Bits	Descriptions
SYSCTL	0x001E	[10:8]	Controls the output slew rate of PIXCLK signal. "111" programs the fastest slew rate. Refer to the AD/CD electrical specification in the data sheet for the slew rate values. A value of "000" produces the slowest slew rate. The default value is "100".
		[6:4]	Controls the output slew rate of GPIO signals. "111" programs the fastest slew rate. Refer to the AC/DC electrical specification in the data sheet for the slew rate values. A default value of "000" produces the slowest slew rate.
		[2:0]	Controls the output slew rate of DOUT[7:0], FV, and LV signals. "111" programs the fastest slew rate. Refer to the AC/DC electrical specification in the data sheet for the slew rate values. A default value of "000" produces the slowest slew rate.

GPIO Control

The MT9T111 has GPIO[3:0] and VGPIO[7:0] for various uses. Both GPIO[3:0] and VGPIO[7:0] are programmable through the two-wire serial interface, to be configured as inputs or outputs. The two ports can also be programmed to provide other functions when they are configured as outputs. Status registers can monitor the values on GPIO[3:0] and VGPIO[7:0] signals, when they function as inputs or outputs.

Overview of GPIO Signals

- GPIO[3:0] and VGPIO[7:0] can be programmed to be either input or output
- GPIO[1:0] can be programmed as the extension of two lower data bits during 10-bit output mode
- GPIO[2] can be programmed as an external shutter control output
- GPIO[3] can be programmed as an external FLASH control output
- GPIO[2] can be programmed as an external /OE control input
- GPIO[3] can be programmed as an external TRIGGER control input
- VGPIO[7:0] can be programmed as a waveform generator output to control external PWM devices

Table 13 shows the related registers to program GPIO signals. Refer to the MT9T111 data sheet for more detail descriptions of the registers and variables.

Table 13: GPIO Related Registers and Variables

Map	Address	Bits	Descriptions
SYSCTL	0x001A	[4]	This bit controls the GPIO ports when the sensor enters standby. 0: GPIOs are powered up in standby 1: GPIOs are powered down in standby
SYSCTL	0x001A	[3]	This bit controls the VGPIO ports when the sensor enters standby. 0: VGPIOs are powered up in standby 1: VGPIOs are powered down in standby
SYSCTL	0x001E	[6:4]	Controls the output slew rate of GPIO signals. “111” programs the fastest slew rate. Refer to the AC/DC electrical specification in the data sheet for the slew rate values. The default value of “000” produces the slowest slew rate.
GPIO	0x0604	[11:8]	GPIO output select register.
GPIO	0x0604	[3:0]	GPIO output register.
GPIO	0x0606	[3:0]	GPIO direction control for each GPIO signal. 0: Input 1: Output
GPIO	0x060A	[3:0]	GPIO input register.
GPIO	0x060C	[15:8]	VGPIO output select register. Set to “1” to enable.
GPIO	0x060C	[7:0]	VGPIO output register.
GPIO	0x060E	[7:0]	VGPIO direction control register. 0: Input 1: Output
GPIO	0x0610	[7:0]	VGPIO input register.

Table 14 describes the GPIO output multiplexer control. Refer to Figure 8 through Figure 12 on page 32 for more details.

Table 14: GPIO Input / Output Multiplexer Control

	Input Mode	Output Mode			
Register Settings	0x0606[3:0] = 0x0	0x0606[3:0] = 0xF 0x0604[11:8] = 0xF	0x0606[2] = 1 0x0604[10] = 0 0x0604[6] = 0	0x0606[3] = 1 0x0604[11] = 0 0x0604[7] = 0	0x0606[1:0] = 11 0x0604[9:8] = 00 0x0604[5:4] = 00 0x0604[13] = 0
GPIO[0]	0x060A[0]	0x0604[0]			DOUT_LSB0
GPIO[1]	0x060A[1]	0x0604[1]			DOUT_LSB1
GPIO[2]	0x060A[2], /OE	0x0604[2]	SHUTTER		
GPIO[3]	0x060A[3], Trigger	0x0604[3]		FLASH	

Figure 8: GPIO[0]

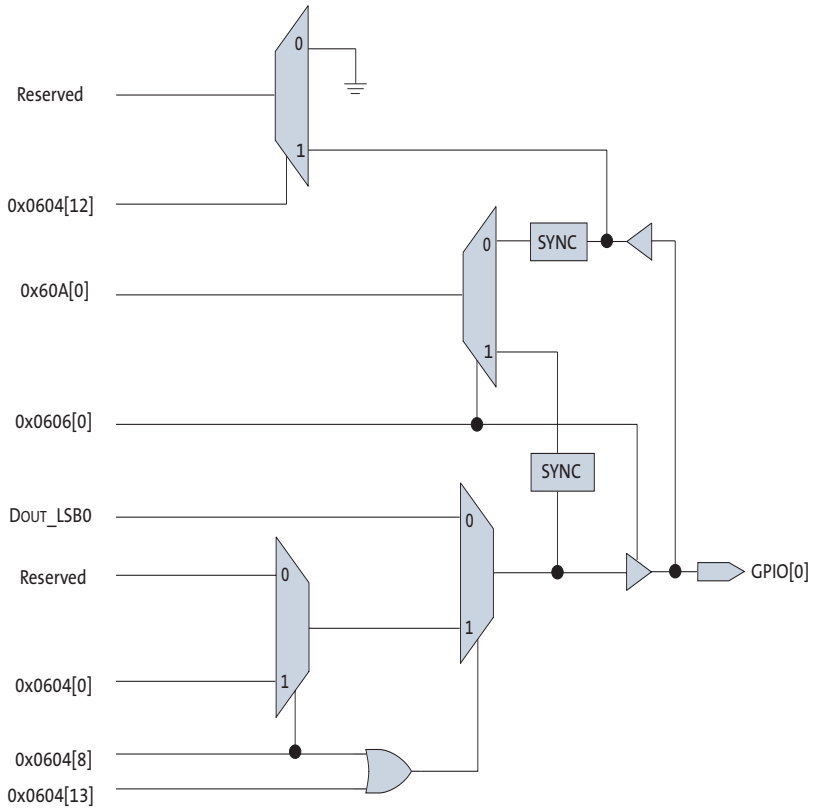


Figure 9: GPIO[1]

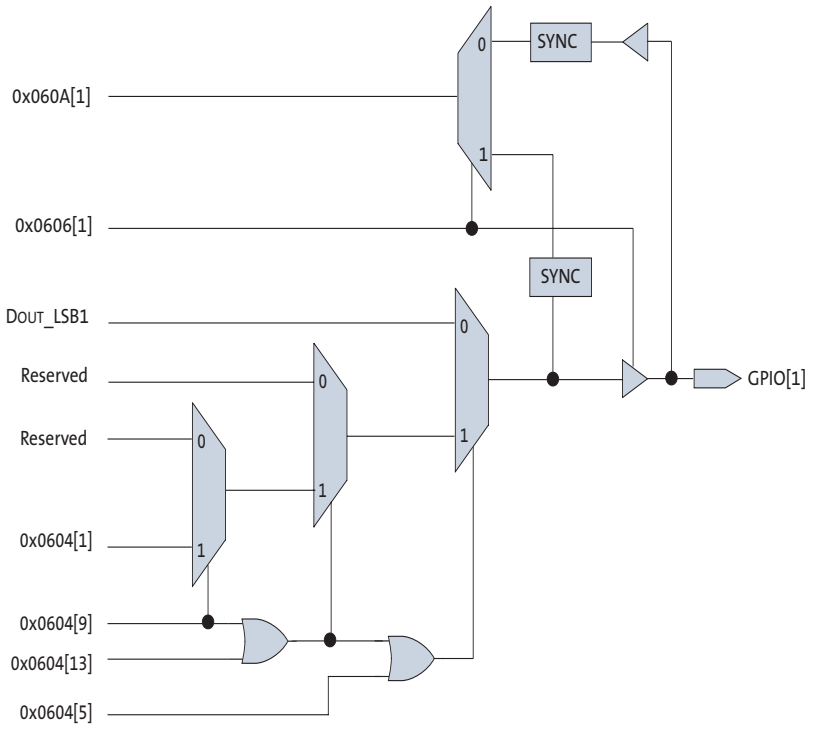


Figure 10: GPIO[2]

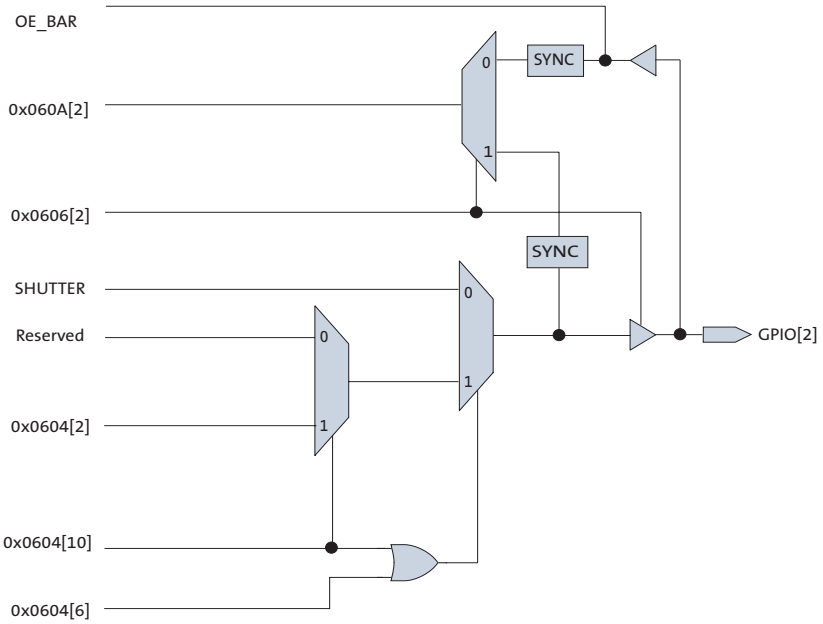


Figure 11: GPIO[3]

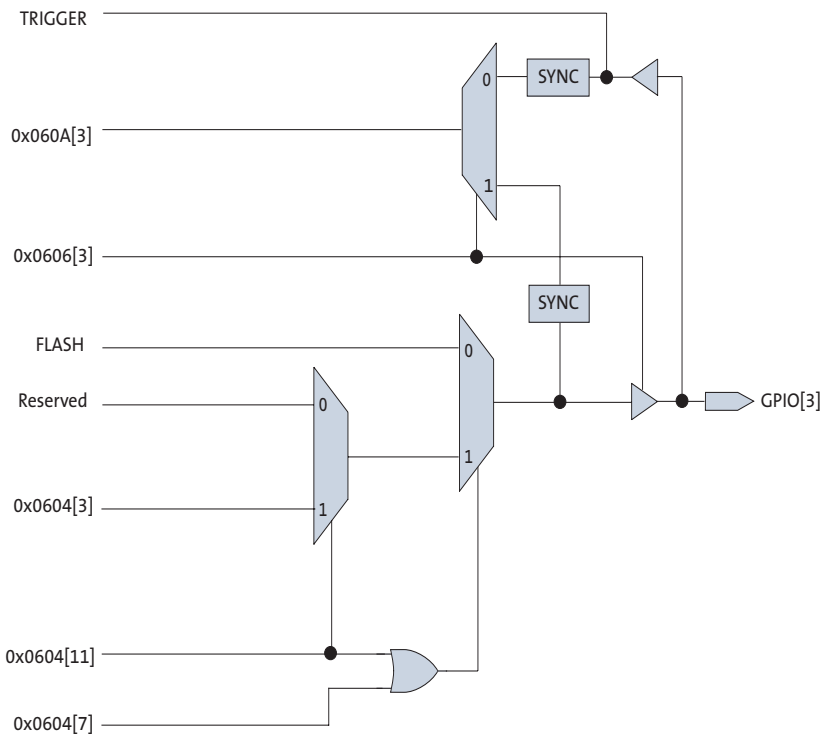
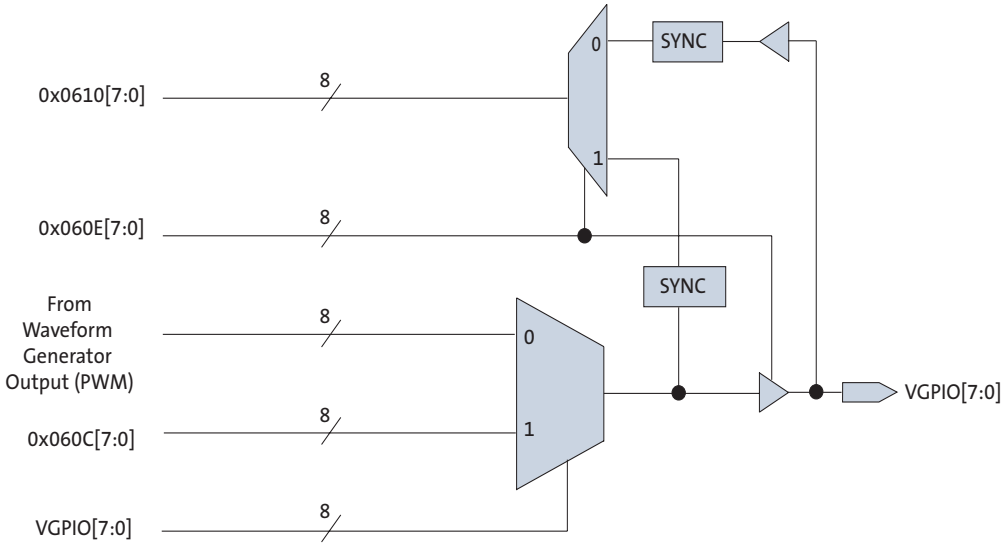


Figure 12: VGPIO[7:0]



One-Time Programmable (OTP) Memory

The MT9T111 Rev3 has 5K-bits of OTP memory that can be used during module manufacturing to store specific module information. This feature enables system integrators and module manufacturers to label and distinguish various module types based on lenses, IR-cut filters, or other properties.

During the programming process, a dedicated pin for high voltage is needed to perform the OTP memory programming. This voltage (VPP) should be 8.5V ±3%. Instantaneous VPP cannot exceed 9V at any time. Completion of the programming process will be communicated by a register through the two-wire serial interface.

Because this programming pin needs to sustain a higher voltage than other input or output pins, having a dedicated high voltage (VPP) pin minimizes the design risk. If the module manufacturing process can probe the sensor at the die or PCB level (that is, supply all the power rails, clocks, and two-wire serial interface signals), then this dedicated high voltage pin does not need to be assigned to the module connector pin out. However, if the VPP signal needs to be bonded out as a pin on the module, the trace for VPP needs to be able to carry a minimum of 4mA for programming only. This pin should be left floating when not programming the OTP memory.

Programming the OTP memory requires the sensor to be fully powered with its clock input applied. After VPP power is applied, the information from the host processor will be programmed through the use of the two-wire serial interface. Once the data is written to internal registers, the host processor sends a program command to the sensor to initiate the programming process.

After the sensor has finished programming the OTP memory, the host machine should poll the write bit through the two-wire serial interface. The write bit will be cleared and go to “0.” Only one programming cycle for the OTP memory can be performed.

Reading the OTP memory data requires the sensor to be fully powered and operational with its clock input applied. The data can be read through registers from the two-wire serial interface.

MT9T111 Rev3 Silicon OTP Memory Programming Procedure

Step 1: Sensor Setup

1. Set V_{AA} = 3.1V and all other supplies to their nominal voltages.
2. Set EXTCLK = 24 MHz.
3. Set the V_{PP} = 8.5V. The supply ramp slew rate should be slow to avoid ESD damage. It is important to take into account the current sourcing capability of the supply so that the voltage does not deviate more than 3% from the target of 8.5V on the V_{PP} pin. At no point should the supply exceed 9V.
4. This example assumes the lens shading correction (LSC) coefficients are being programmed into the OTP memory. The same method can also be employed to program a module ID or other miscellaneous settings that do not require firmware to execute the settings.

Step 2: Initialize the Sensor for OTP Memory Programming

1. WRITE R0x0010 = 0x0008
2. WRITE R0x0012 = 0x00F1
3. WRITE R0x002A = 0x73DA
4. WRITE R0x0014 = 0x2545
5. WRITE R0x0014 = 0x2547
6. WRITE R0x0014 = 0x3447
7. WRITE R0x0014 = 0x3047
8. POLL R0x0014[15] until its value = 1 // PLL locked
9. WRITE R0x0014 = 0x3046
10. WRITE R0x0016 = 0x0400 // JPEG initialization workaround
11. WRITE R0x0030 = 0x0003
12. WRITE R0x0018 = 0x402C
13. POLL R0x0018[B14] until its value = 0
14. WRITE R0x3812[11:7] = 0x05 // Comparator is 5
15. WRITE VAR = 24, R0x0037[0] = 0x0001//Probe the OTP memory
16. POLL VAR8 = 24, R0x0039[0] until its value = 1 // OTP memory detected

Step 3: Programming the Data

1. WRITE VAR = 24, R0x0031 = 0x0 // Present address
2. WRITE VAR = 24, R0x0033 = 0xAAAA // Present dummy data
3. WRITE VAR = 24, R0x0037 = 0xA080 // Execute write word command
4. POLL VAR = 24, R0x0037[7] until its value = 0 // Wait for Bit 7 to clear indicating the WRITE has completed
5. WRITE VAR = 24, R0x0031 = 0x2 //Address increments by two
6. WRITE VAR = 24, R0x0033 = DATA1 // Data1 programmed to PGA1
7. WRITE VAR = 24, R0x0037 = 0xA080 // Write word
8. POLL VAR = 24, R0x0037[7] until its value = 0
9. WRITE VAR = 24, R0x0031 = 0x4 // Address increments by two
10. WRITE VAR = 24, R0x0033 = DATA2 // Data2 programmed to PGA2
11. WRITE VAR = 24, R0x0037 = 0xA080 // Write word
12. POLL VAR = 24, R0x0037[7] until its value = 0
13. WRITE VAR = 24, R0x0031 = 0x6 // Address increments by two
14. WRITE VAR = 24, R0x0033 = DATA3 // Data3 programmed to PGA3
15. WRITE VAR = 24, R0x0037 = 0xA080 // Write word

16. POLL VAR = 24, R0x0037[7] until its value = 0
17. Repeat the procedure above, incrementing the address until all the LSC coefficients are programmed.

Notes:
1. In the example above only three LSC coefficients (DATA1–3) are programmed.
2. Address 0x0 should be programmed with a dummy value. An LSC coefficient (DATA) should not be programmed into it.
3. Programming should always be done in sets of two DATA points.
   Example: In Steps 9–16 two data points, DATA2 and DATA3 were programmed.
4. While reading the LSC data from OTP memory into the PGA table (SOC2 register), a starting register of 0x2 should be used, since it's the first valid LSC coefficient (DATA).

Device ID

The MT9T111 provides device ID for identifying the sensor after power-up by the host processor.

Table 15: Device ID Related Registers and Variables

Map	Address	Bits	Descriptions
SYSCTL	0x0000	[15:0]	Contains the MT9T111 device ID number, 0x2680. Read-only.

Module ID

The MT9T111 provides two ways to implement module ID using GPIO ports or OTP memory.

1. GPIO[3:0] states are sampled during power-on and stored in registers located in 0x3026[3:0]. External pull-up or pull-down resistors are required to set each signal to a valid logic state. External resistors should be pulled up to the VDD_IO source. The MT9T111 samples GPIO[3:0] during the end of the reset cycle to ensure a valid state when VDD and VDD_IO come from the same power supply.
2. The OTP memory must be programmed during the manufacturing process. Rev3 silicon provides up to 5K-bits of data for module ID and other module variation data, for example, position-dependent gain adjustment (PGA) coefficients for lens shading correction.

Refer to the "One-Time Programmable (OTP) Memory" on page 33 for programming and verification.

Master Two-Wire Serial Interface and External Sensor Control Interface

The MT9T111 has one slave and one master two-wire serial interface. The slave two-wire interface communicates with the host system to program internal registers and variables, just as with other Aptina sensor products. The MT9T111 has a master two-wire interface including an AF driver, which can control an external secondary sensor and other I^2C devices. The MT9T111 also generates the system signals for the external secondary sensor including reset, standby, and clock. Table 16 shows the registers related to controlling the external secondary sensor.

Table 16: Control of an External Secondary Sensor

Map	Address	Bits	Descriptions
GPIO	0x0600	[0]	External secondary sensor RESET signal. Active High.
GPIO	0x0600	[1]	External secondary sensor STANDBY signal. Active High.
GPIO	0x0602	[2:0]	The OE signals for secondary sensor controls. 001: Output enable for secondary sensor control RESET 010: Output enable for secondary sensor control STANDBY 100: Output enable for secondary sensor clock
GPIO	0x0614	[0]	Master two-wire serial interface SDA signal enable. Active LOW. Default for SDA signal is power off.
SYSCTL	0x002E	[10]	External secondary sensor clock source select. 0: Use Output of P7 divider from using PLL 1: Use EXTCLK
SYSCTL	0x002E	[8]	External secondary sensor clock enable. 0: Disable 1: Enable
SYSCTL	0x002E	[3:0]	External secondary sensor *e* divider value. Actual *e* divider value will be the programmed [3:0] * 2. For example, if [3:0] is programmed as 0x2 then e is 4. Divider value of *e* is from 1 ([3:0] = 0x0) to 32 ([3:0] = 0xF). External secondary sensor clock source will be divided by e to generate final external clock value. External secondary sensor clock = (EXTCLK or P7 divider output) / *e*.
SYSCTL	0x0032	[15:0]	Master two-wire serial interface clock slew rate control.

Output Interface

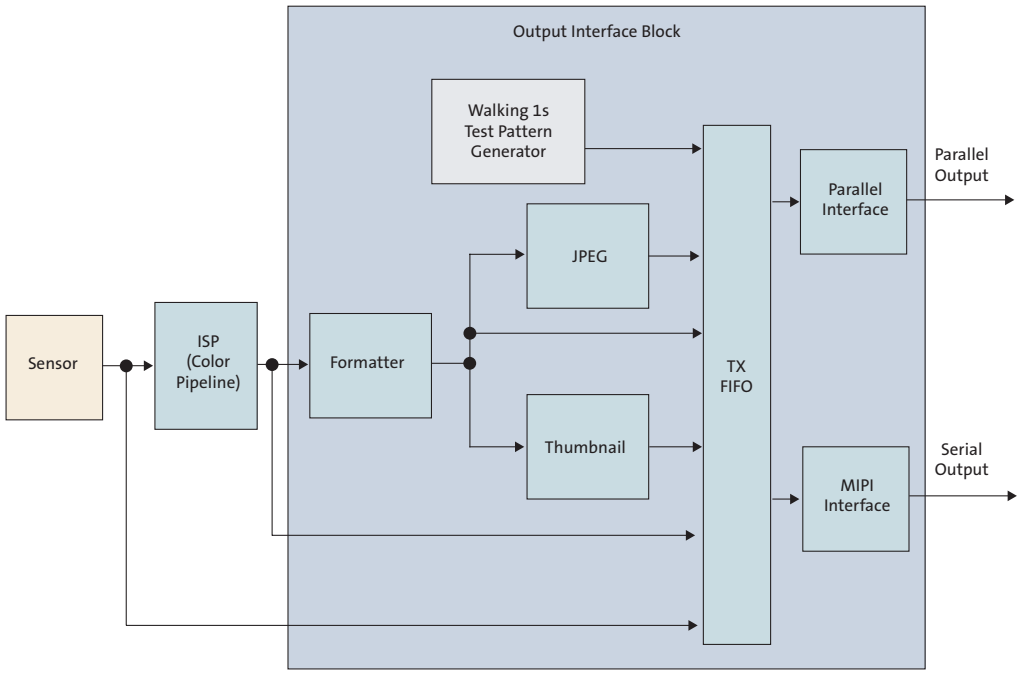
This section describes how to interface the MT9T111 to the host controller. The following topics are discussed:

- JPEG encoder
- Thumbnail
- Output data format
- Walking 1s test pattern generator

The MT9T111 has flexible configuration options to adjust the output data and blanking timing. It may be configured to have fixed timing or scene-dependent variable timing. Therefore, the host controller must synchronize using FV and LV signals and latch output data only when FV = 1 and LV = 1.

The MT9T111 provides two external data ports—parallel and MIPI serial. Only one interface type can be programmed at a time. The output data multiplexer block provides several output data path options as shown in Figure 13.

Figure 13: Output Interface Block



JPEG Encoder

The JPEG compression engine in the MT9T111 is a highly integrated, high-performance solution that provides for low power consumption and full programmability of JPEG compression parameters for image quality control.

The JPEG encoding block is designed for continuous image flow and is ideal for low power applications. After initial configuration for a target application, it can be controlled easily for instantaneous stop or restart. A flexible configuration and control interface allows for full programmability of various JPEG-specific parameters and tables.

JPEG Encoding Highlights

- Sequential DCT (baseline) ISO/IEC 10918-1 JPEG-compliant
- YCbCr 4:2:2 and 4:2:0 format compression
- Support for two pairs of programmable quantization tables
- Quality/compression ratio control capability
- 15 fps JPEG capability at full resolution (header processing in external host processor)
- Support for interleaved RGB thumbnail up to 640 x 480
- Capture color pipe bypass stream (10-bit), JPEG bypass stream (16-bit), scalar bypass stream (16-bit) or JPEG encoded stream (8-bit), as programmed by host or microcontroller
- JPEG encoded stream can work in continuous mode or spoof mode
- JPEG encoded stream working in continuous mode can only transmit on the parallel output port
- SOI/EOI can be inserted before or after JPEG encoded stream working in continuous mode at the parallel output port
- Thumbnail can be enabled for the JPEG encoded stream in both continuous and spoof mode
- In spoof mode, data is output with programmed frame timing; dummy patterns may be padded as necessary
- In spoof mode, fill patterns may be padded when the thumbnail is enabled, and thumbnail data is not long enough to fill one line
- Spoof-frame height can be ignored in spoof mode

JPEG Output Interface

JPEG Data

JPEG data can be output in both the parallel and the serial MIPI streams. In the parallel output interface, JPEG data is output on the 8-bit parallel bus DOUT[7:0], with FV, LV, and PIXCLK. JPEG output data is valid when both FV and LV are asserted. When the JPEG data output for the frame completes, or buffer overflow occurs, LV and FV are de-asserted.

The MT9T111 can transmit JPEG data using two different formats: JPEG continuous stream and JPEG spoof stream. In both formats, JPEG status segments containing information (resolution, file size, and status) about the image and the offsets of thumbnail data can be inserted into the output streams. The following sections describe the two streaming methods.

RGB Thumbnail

To support display of captured images without decoding a JPEG file, the MT9T111 can output a resized version of the captured JPEG data as an RGB thumbnail image embedded in the JPEG stream.

This thumbnail image is computed from the same image that is input to the JPEG compressor, and is scaled to a user-programmable size, from 160 x 120 to 640 x 480. The thumbnail size must be configured to be smaller than the JPEG image size.

This image can be separated by parsing the stream for tags surrounding the embedded image. Alternatively, the embedded image can be extracted without parsing by reading thumbnail data offsets from the thumbnail pointer table. This thumbnail pointer table is optionally output in the image status segment, and contains one entry for each line of thumbnail data.

JPEG Continuous Stream

JPEG continuous stream goes out only through the parallel output interface, and supports the following features:

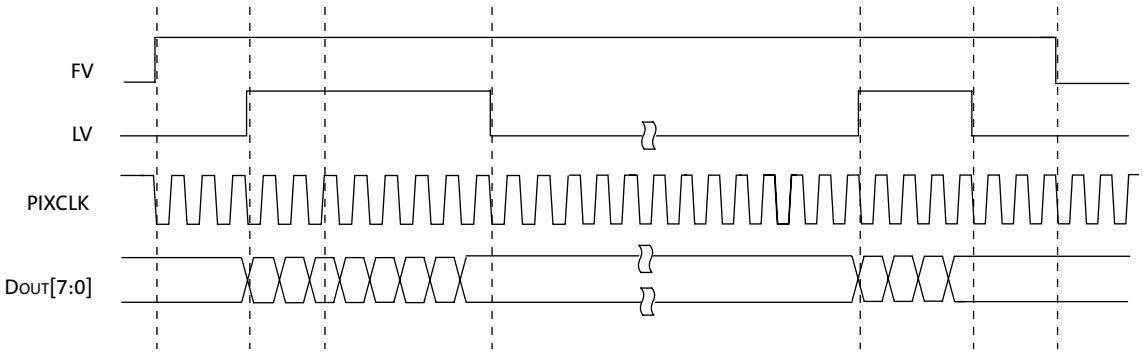
- Adaptive clock switching
- Insertion of Start of Image (SOI) and End of Image (EOI) codes
- Duplicate FV on LV
- Append JPEG status segment at the end of the data stream

When enabled, the pixel clock output can be generated continuously during invalid data periods (between FV and between LV). In this streaming mode, the amount of valid data within each line (LV = 1) is variable. When adaptive clock mode is enabled, the pixel clock is adjusted to lower clock rates, based on the fullness of the output FIFO. Figure 14 through Figure 18 on page 44 are examples of the JPEG stream through the parallel output interface.

Figure 14 illustrates data output when the pixel clock output is generated continuously during invalid data periods. LV is of variable length based on data output rate.

In default mode for Rev3 silicon, data transitions on the falling of PIXCLK and the host must capture data on the rising edge of PIXCLK. The PIXCLK is also configurable and its polarity can be reversed through the use of R0x3C20 (refer to Table 8 on page 21).

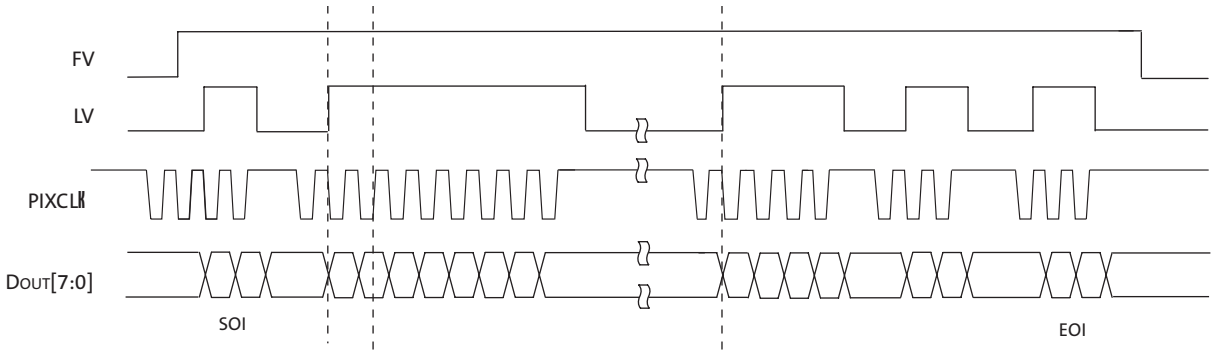
Figure 14: JPEG Continuous Data Output



Notes:
1. Under default conditions FV and LV are asserted on the falling edge of PIXCLK.
2. Data must be captured by the host on the rising edge of PIXCLK.

Figure 15 illustrates when SOI and EOI codes are inserted in the parallel output interface.

Figure 15: JPEG SOI and EOI Inserted



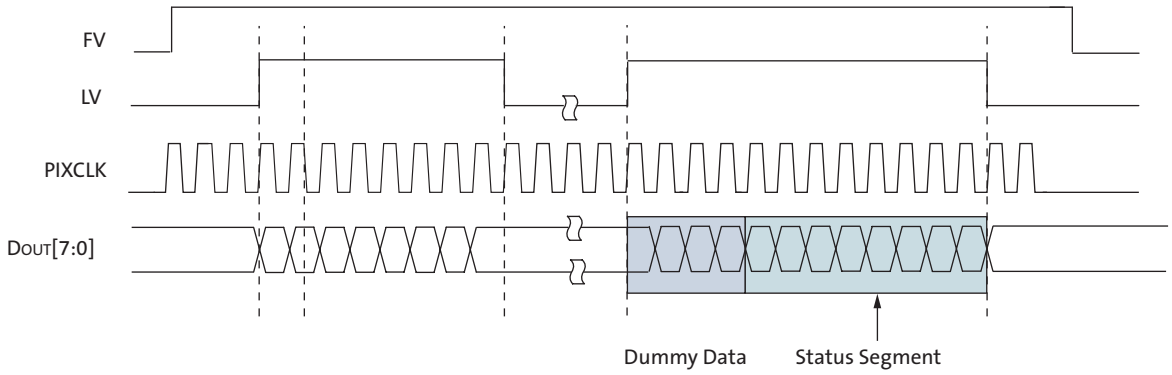
JPEG Spoof Stream

The JPEG compressed data can be output in spoof mode. In spoof mode, there is no SOI/EOI option available. The amount of expected pixel data is defined by the width and height registers in spoof mode. If the valid JPEG data is less than expected size defined, a register-programmable dummy data pattern with a default value of 0xFF will be padded. It is mandatory that the last 6 bytes contain 2 bytes of status, followed by 4 bytes of length information.

When enabled, the pixel clock output can be generated continuously during invalid data periods (between FV and between LV). In this streaming mode, the amount of valid data within each line (LV = 1) is constant. When adaptive clock mode is enabled, the pixel clock is readjusted to lower clock rates, based on the fullness of the output FIFO. Below are some examples of the JPEG spoof stream.

Figure 16 illustrates the JPEG spoof output when pixel clock is generated continuously during invalid data periods between LV. The status segment is inserted at the end of the stream.

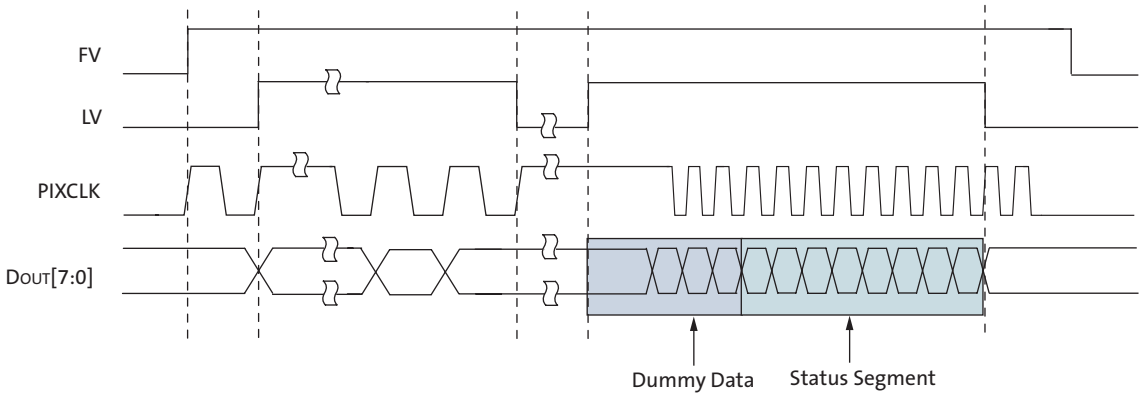
Figure 16: JPEG Spoof Mode Timing with Continuous Clock



Notes: 1. PIXCLK is reversed in this example with data output on the rising edge of PIXCLK and data captured by the host on the falling edge of PIXCLK.

Figure 17 illustrates the JPEG spoof output when the adaptive clock mode is enabled. With continuous PIXCLK, the switching of the PIXCLK frequency can happen at any time.

Figure 17: JPEG Spoof Mode Timing with Adaptive Clock



Notes: 1. PIXCLK is reversed in this example with data output on the rising edge of PIXCLK and data captured by the host on the falling edge of PIXCLK.

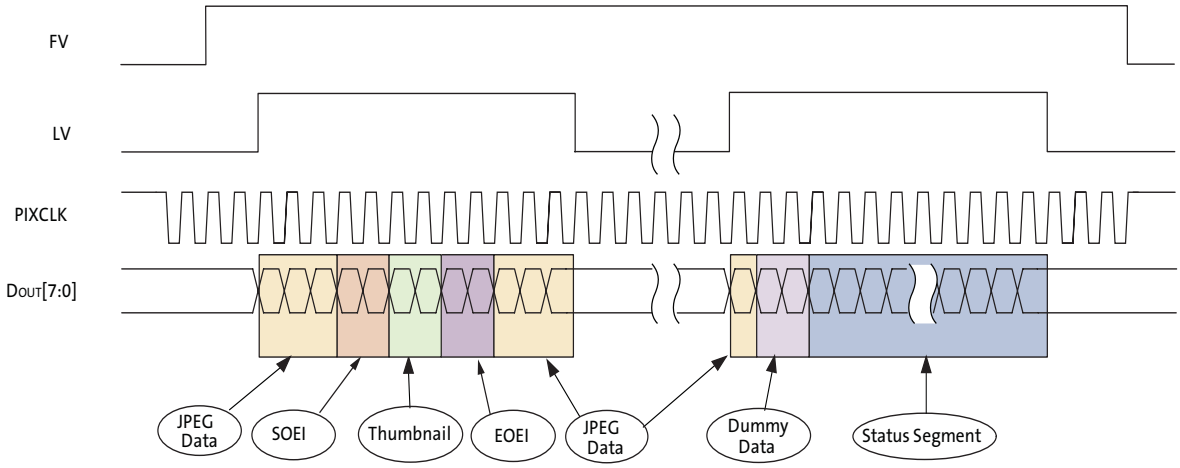
JPEG Spoof Stream in MIPI Output Mode

In MIPI output mode, only the JPEG spoof stream can be output. Similar to the parallel output interface, the amount of expected pixel data is defined by the width and height registers in spoof mode. If the valid JPEG data is less than expected size defined, register-specified dummy data will be padded.

JPEG Stream with Embedded Thumbnail Image

In JPEG mode, it is possible to embed a scaled uncompressed image to the compressed data stream. This image is interleaved within the data (as shown in Figure 18), and must be separated before saving the compressed image. The embedded image is separated from the main image by Start of Embedded Image (SOEI) and End of Embedded Image (EOEI) tags. These tags are register-programmable codes that enable a host to parse the thumbnail data from the compressed image stream.

Figure 18: JPEG Spoof Mode Timing with Thumbnail



Notes:
1. PIXCLK is inverted in this example.
2. Thumbnail start and end codes are programmable by register setting.
3. Status segment includes JPEG pointer table.

In addition, the output formatter can append a table of thumbnail data offsets to the status segment of the image. This thumbnail index pointer shall have one entry for each line of thumbnail data. Each entry is a 4-byte pointer containing the offset of the valid thumbnail data.

Transfer Modes

Two JPEG transfer modes are supported for different output streams:

- Continuous mode
- Spoof mode

The output buffer module takes inputs from different sources—the color pipe bypass stream, the JPEG bypass stream, and the JPEG compressed stream. Each data stream source may have its own transfer mode, which is shown in Table 17.

Table 17: Transfer Modes and Sources

Source/Mode	Bypass Mode R0x3C52[1:0] = "10"	Continuous Mode R0x3C52[1:0] = "00"	Spoof Mode R0x3C52[1:0] = "01"
Color pipe bypass stream R0x3C50[1:0] = "10"	X		
JPEG bypass stream R0x3C50[1:0] = "01"	X		
JPEG compress stream R0x3C50[1:0] = "00"		X	X

Bypass Mode

In this scenario, the hardware ignores the option of changing the clock speed depending on data rates. The clock remains at the frequency of PCLK1. However, the software can slow down PCLK1 by programming the output buffer PCLK1 configuration register. It is the responsibility of the user to guarantee that the buffer does not overflow or underflow.

CCIR markers can be optionally inserted. However, because the markers SOF and EOF are carried on data before FV is asserted, and after FV is asserted; SOL and EOL markers are carried on data before LV is asserted and after LV is de-asserted, CCIR markers can NOT be guaranteed to be transferred on the MIPI/CCP interface. This feature only applies to the parallel output port.

Continuous Mode

The JPEG output clock can be configured to be either continuous or gated off while LV is de-asserted. The JPEG output clock can also be optionally gated off between frames to save power.

The JPEG data stream implements two markers, SOI (0xFFD8) and EOI (0xFFD9), which can be optionally inserted before and after valid JPEG data, respectively. SOI and EOI can be inserted either inside or outside the FV assertion period, but always inside the LV assertion period.

The adaptive clock switching feature is also supported. The PCLK changes according to the output buffer FIFO fullness status.

When the thumbnail asserts a request and the FIFO is less than 5/8 full, the output buffer serves the thumbnail immediately until the end of the thumbnail line. Then the output buffer switches back to JPEG data if there is any. When the thumbnail asserts a request and the FIFO is more than 5/8 full, JPEG data has higher priority. The output buffer does not service the thumbnail request until the FIFO drops below 5/8 full.

JPEG status information can be appended at the end of JPEG/Thumbnail stream.

Spoof Mode

For spoof mode, the LV signal is asserted and de-asserted in a pattern intended to emulate the behavior of multiple horizontal video blanking intervals.

The software configures the spoof pattern by programming the total number of LV assertion intervals, as well as the number of output clock periods both during and between LV assertions. The user can configure a virtual JPEG output frame that is specifically tailored to the expected JPEG file size. If this frame is larger than the total number of JPEG bytes actually produced, the output buffer pads the remaining bytes of the last JPEG data line with dummy patterns, then either de-asserts FV or continues to output dummy patterns until end of frame. If the frame is too small, output buffer either continues to output the excess JPEG bytes until whole JPEG frame is output or discards the excess JPEG bytes and sets an error flag in a status register.

Note: Although the number of bytes on each JPEG line will be same, the corresponding time interval will vary due to instantaneous changes in the JPEG data rate.

For the parallel output interface, when the FIFO underflow happens during one line transmit, the PCLK will be gated off and wait for more data. For the MIPI/CCP interface, the output buffer will not start transmitting until one line of data is ready in the FIFO.

The JPEG output clock can be configured to be either gated off or continuous while LV is de-asserted. The JPEG output clock can also be optionally gated off between frames to save power.

The adaptive clock switching feature is supported for the parallel output interface. The PCLK changes according to the output buffer FIFO fullness status.

The 2 bytes of JPEG status information and 3 bytes of JPEG length information can be enclosed with SOSI/EOSI codes, and appended to be the last 9 bytes of the last line in the frame.

When the thumbnail asserts a request and the FIFO is less than 5/8 full, the output buffer services the thumbnail request immediately until the end of thumbnail line. Then the output buffer switches back to JPEG data, if there is any. When the thumbnail asserts a request and the FIFO is more than 5/8 full, JPEG data has higher priority. The output buffer doesn't service the thumbnail request until the FIFO drops below 5/8 full.

Thumbnail Index Table

Some customers may use processors which are not efficient in pattern search operations, therefore their systems take more time to extract thumbnail data from the JPEG stream by looking for thumbnail header and footer code set. The thumbnail index table provides an alternative solution to speed up the thumbnail extraction.

The position (pointer) of the first thumbnail data for every thumbnail line is captured by the sensor. These pointers are saved in the internal memory during transmission, and can be appended at the end of the frame so that the user can decode these pointers and locate the thumbnail data position directly.

Thumbnail Index Pointer

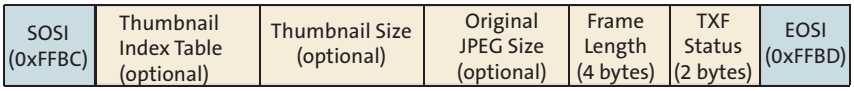
The thumbnail index pointer is the content to be stored in the thumbnail index table, and every pointer consumes one entry in the table.

Note: The index pointer is defined as the first data of every thumbnail line, which does not account the SOEI. For example, if the first line of thumbnail data is placed at the beginning of the outgoing data stream, the first thumbnail index pointer (P0) is 0x0002.

JPEG Status Segment

To provide the user quick knowledge of the status when the JPEG plus thumbnail is enabled, a JPEG status segment is appended at the end of frame. This segment is optional in continuous mode, while it is mandatory for spoof mode.The status segment is enclosed by SOSI/EOSI codes, as shown in Figure 19.

Figure 19: Contents of Status Segment

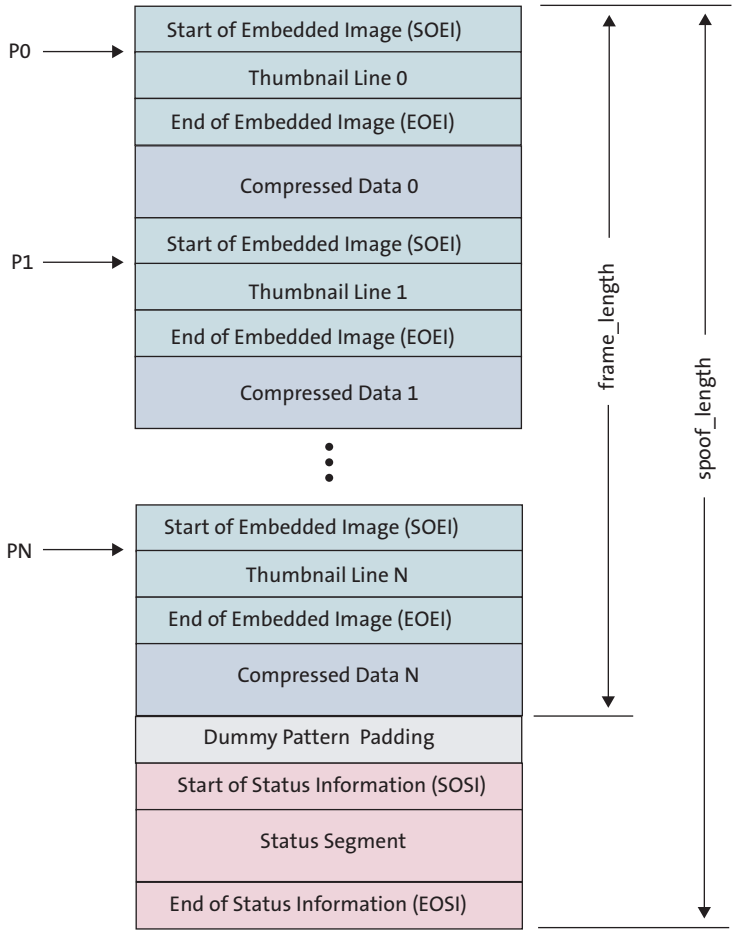


The contents of the status segment are summarized as follows:

- SOSI, start of status information, which is coded as 0xFFBC
- Thumbnail index table (every entry has 4 bytes) is asserted and thumbnail is enabled
- The width of thumbnail in pixels (2 bytes)
- The height of thumbnail (2 bytes)
- The width of uncompressed full image
- The height of uncompressed full image
- 4-byte JPEG plus thumbnail length
- 2-byte status
- EOSI, end of status information, which is coded as 0xFFBD

Either thumbnail data or JPEG data starts first, depending on the time of their availability.

Figure 20: JPEG Data Segment Structure



Utilization of the Thumbnail Index Pointer

The number of thumbnail index pointers is the same as the number of thumbnail lines per frame. Every pointer consists of 4 bytes, which are transmitted from MSB to LSB byte.

This portion is only available when the thumbnail is enabled. For the current version of the MT9T111, only continuous mode has implemented the thumbnail index table logic.

Original JPEG and Thumbnail Image Resolution

The synchronization between DevWare and firmware has been a critical issue. Once DevWare changes either JPEG or thumbnail resolution, the firmware is responsible for configuring the hardware registers. However, DevWare has no idea when the resolution change takes effect, or when it processes the received frames.

The solution to this unknown timing is to append the uncompressed image resolution and the thumbnail resolution in the status segment, so that Devware is able to extract resolution information for every received frame. The resolution field consists of two bytes of width and two bytes of height (big endian) of the uncompressed JPEG frame. If the thumbnail is enabled, the size of the thumbnail will also be inserted before the thumbnail.

In summary, the resolution field is shown in Table 18, and every part is MSB first.

Table 18: Resolution Field

Data Type	Size
Thumbnail width in pixels	2 bytes
Thumbnail height	2 bytes
Uncompressed JPEG image width	2 bytes
Uncompressed JPEG image height	2 bytes

The resolution field will be inserted in the status segment. If the thumbnail is disabled, the thumbnail width and thumbnail height sections will be filled with 0x0000.

JPEG and Thumbnail Length Information

The JPEG length information consists of 4 bytes (the frame length). If the JPEG stream does not contain a thumbnail, it's length is the length of the JPEG compressed data. If the JPEG stream includes a thumbnail, it's length is the length of the JPEG compressed data plus the length of the thumbnail.

JPEG Status Information

In JPEG continuous mode and JPEG spoof mode, two bytes of JPEG status may be inserted at the end of data stream for the user's convenience.

JPEG status consists of two bytes, which carries the information, as shown in Table 19.

Table 19: JPEG Status Description

Status	Field Name	Bits	Description	Default
status 0 byte	thumbnail_fifo_ovf	7	Thumbnail FIFO overflow status.	0
	thumbnail_fifo_udf	6	Thumbnail FIFO underflow status.	0
	reserved	5	Reserved.	0
	thumbnail_frame_ovf	4	Thumbnail frame overflow status.	0
	reserved	3	Reserved.	0
	which_qtable	2:1	It tells which Q-table is used for JPEG core.	0
	rb_error	0	If asserted, it indicates the reorder buffer error occurs in current frame.	0
status 1 byte	frame_rcv_done	7	Frame receive done. If asserted, it indicates that the current frame has been received by output buffer.	0
	fifo_watermark[1:0]	6:5	Watermark of output buffer FIFO usage for this frame. 11: Indicates highest utilization of output buffer FIFO is greater than 75%, but less than 100%. 10: Indicates highest utilization of output buffer FIFO is less than 75%. 01: Indicates highest utilization of output buffer FIFO is less than 50%. 00: Indicates highest utilization of output buffer FIFO is less than 25%.	0
	frame_ovf	4	The frame overflow status. If asserted, it indicates the previous frame has not finished when a new frame comes in.	0
	spoof_ovsize	3	Spoof oversize error status flag. When asserted, it indicates that the spoof frame size is too small for the JPEG encode stream.	0
	Reserved	2	Reserved	0
	obfifo_ovf	1	Output buffer FIFO overflow status. When asserted, it indicates that an overflow condition was detected in the output buffer FIFO during the frame transfer and that transfer was terminated prematurely.	0
	Reserved	0	Reserved.	0

Notes: 1. The status 0 byte is transmitted first, followed by the status 1 byte.

Error Handling

FIFO Underflow

FIFO underflow only occurs in bypass mode, and when it occurs, one output line may split into two or more lines. However, all of the data will be sent out when they are available.

The user can use the FIFO length spreadsheet to calculate appropriate trigger marks to avoid FIFO underflow errors.

FIFO Overflow

FIFO overflow may happen in bypass, continuous, and spoof mode. This error triggers a FIFO flush and results in early termination of the frame.

To avoid a FIFO overflow in bypass mode, use the FIFO length spreadsheet to calculate appropriate trigger marks to avoid FIFO overflow errors.

In continuous and spoof mode, the following factors may trigger a FIFO overflow:

- The output pixel clock slowed down by programming PCLK1, PCLK2, and PCLK3 divider registers, while the JPEG data amount is large
- A decrease in the horizontal blanking time
- Thumbnail mode is enabled, which consumes some time

Frame Overflow

Frame overflow may happen in bypass, continuous, and spoof mode. This error triggers a FIFO flush and results in early termination of the frame.

To avoid frame overflow in bypass mode, the user shall take the following factors into account:

- TXSS input horizontal blanking time
- TXSS input vertical blanking time
- Output pixel clock speed

In continuous and spoof mode, the following factors may trigger frame overflow:

- TXSS input horizontal blanking time
- TXSS input vertical blanking time
- Output pixel clock slowed down by programming PCLK1, PCLK2, and PCLK3 divider registers, while the JPEG data amount is large
- Thumbnail mode is enabled, which consumes some time
- The thumbnail index table is enabled, which consumes some time

Spoof Oversize Error

The spoof oversize error only occurs in spoof mode running on the parallel output interface (the MIPI/CCP interface does not have a spoof oversize error). This error occurs when the spoof size, which is equal to spoof_width * spoof_height is not long enough to hold the compressed JPEG data, thumbnail data (if available), and the status segment. When it occurs in one frame, the thumbnail table will be the first one to be dropped. If the spoof size is still not large enough to hold the data, compressed JPEG or thumbnail data will be truncated so that the rest of status segment (excluding the thumbnail table) is completed for the user's reference.

To avoid a spoof oversize error, the user has two alternatives:

- Assert spoof height ignore option
- Increase spoof width or spoof height

Parallel Output Interface

Protocol

The parallel output interface consists of the following signals.

- FV
- LV
- DOUT[7:0]
- PIXCLK
- pads_slew_rate

JPEG data or raw data are output on an 8-bit parallel data port DOUT[7:0] with the frame valid (FV) signal to qualify JPEG frame timing, the data valid (LV) signal to qualify valid data, and the output clock (PIXCLK). pads_slew_rate is used to control switching speed of the clock signals.

Bypass mode, continuous mode, and spoof mode can run on the parallel output interface.

Features

The following features are implemented for the user's convenience:

- PCLK can be turned on or off during frame blanking, which is configured by register bit tx_control.en_clk_between_frames.
- In bypass or spoof mode, PCLK can be turned on or off during line blanking, which is configured by register bit tx_control.en_clk_between_lines.
- In continuous mode, PCLK can be turned on or off when data is not valid, which is configured by register bit tx_control.en_clk_invalid_data.
- In continuous mode, SOI or EOI can be inserted.
- In bypass mode, CCIR markers can be inserted, instead of po_fvld and po_dvld to determine fame valid and line valid.
- Adaptive clock switching is supported on the parallel output interface.

Adaptive Clock Switching

The adaptive clock switching feature is implemented on the parallel output interface for continuous and spoof mode, but is not supported for bypass mode.

When ob_tx_control.en_adaptive_clk is enabled, the hardware automatically switches clocks according to FIFO fullness status.

Three clock dividers are implemented for PCLK1, PCLK2, and PCLK3 in their associated configuration registers (0x3C66, 0x3C88, 0x3C8A). To utilize this feature, PCLK1 should be configured as the slowest clock (for example, divider = 6), PCLK2 should be the second slowest (for example, divider = 3), and PCLK3 should be the fastest clock (for example, divider = 1).

After reset, the output PCLK stays at PCLK1. It switches to PCLK2 when the FIFO reaches 50 percent full, switches to PCLK3 when FIFO reaches 75 percent full, or to PCLK1 when FIFO drops below 25 percent full. If the output buffer is currently using PCLK3, it will switch to PCLK2 when the FIFO is below 50% full. The switching criteria is summarized in Table 20.

Table 20: Clock Switching Criteria

Fullness/Selection	PCLK1	PCLK2	PCLK3
25%	Stay at PCLK1	Switch to PCLK1	N/A
50%	Switch to PCLK2	Stay at PCLK2	Switch to PCLK2
75%	N/A	Switch to PCLK3	Stay at PCLK3

Output Interface Timing

This section outlines the outgoing signal timing in all different cases.

Parallel Output Interface

JPEG bypass stream, color pipe bypass stream, JPEG continuous stream, and JPEG spoof stream can be exclusively selected on the parallel output interface. However, different streams have their own outgoing formats, which are shown in the following sections.

Notes:
1. The outgoing PCLK can be inverted if the user chooses to do so.
2. PIXCLK is configurable and can be inverted through the use of R0x3C20.

Summary of Parallel Output Interface Options

The output buffer supports many output options, as shown in Table 21. The settings in the table are configurable through:

VAR = 26, 0x00A0, 0x082E // PRI_A_CONFIG_JPEG_OB_TX_CONTROL_VAR
VAR = 27, 0x00A0, 0x082E // PRI_B_CONFIG_JPEG_OB_TX_CONTROL_VAR

Table 21: Parallel Output Interface Options

Options/Mode	Description	Case	Bypass Mode	Continuous Mode	Spoof Mode
Thumbnail	Thumbnail stream only applies to JPEG compressed stream.	Continuous: 7 Spoof: 3, 4, 5, 6	N/A	Support	Support
en_ccir_code	CCIR markers only apply to bypass mode.	Bypass: 4	Support	N/A	N/A
en_clk_invalid_data	0: PCLK is only enabled when data is valid. 1: PCLK is a free running clock when FRAME_VALID is asserted	Cont: 1, 2, 3, 4, 5, 6, 7, 8, 9	N/A	Support	N/A
en_clk_between_frames	0: Turn off clock during frame blanking. 1: Turn on clock during frame blanking.	All Cases	Support	Support	Support
en_clk_between_lines	0: Turn off clock during line blanking. 1: Turn on clock during line blanking.	Bypass: 1, 2, 3, 4 Spoof: 1, 2, 3, 4, 5, 6	Support	N/A	Support
en_adaptive_clk	Enable adaptive clock switching. It does not apply to bypass mode.	Cont: 8, 9 Spoof: 6	N/A	Support	Support
en_soi_eoi	When asserted, insert an SOI at the beginning of the JPEG stream, and an EOI at the end of the JPEG stream.	Cont: 5, 6, 7	N/A	Support	N/A
soi_eoi_in_fv	This bit is valid only when EN_SOI_EOI is set. 0: SOI is asserted before FV, and EOI is asserted after FV. 1: SOI is asserted after FV, and EOI is asserted before FV. Basically, they are within FV being asserted.	Cont: 5, 6, 7	N/A	Support	N/A
insert_jpeg_status	This is an option for continuous mode. However, it is mandatory for spoof mode.	Cont: 6, 7	N/A	Support	N/A
dup_fv_on_lv	The user can always turn on this option if the user's microcontroller has the bandwith and capability.	Cont: 3	Support	Support	Support

Table 21: Parallel Output Interface Options (continued)

Options/Mode	Description	Case	Bypass Mode	Continuous Mode	Spoof Mode
en_byte_swap	When set, it enables the swapping of the byte order between a pair of data bytes. The user can always turn on this option if the user's microcontroller has the bandwith and capability.		Support	Support	Support

JPEG Bypass Stream and Color Pipe Bypass Stream

The JPEG bypass stream and color pipe bypass stream are from different sources. Since their output formats are the same, they can be classified as the same category.

Bypass streams do not support adaptive clock switching (it remains PCLK1) because the incoming throughput is fixed. However, the user can use a slower output clock by programming the PCLK1 configuration register. The user can also choose to insert CCIR codes.

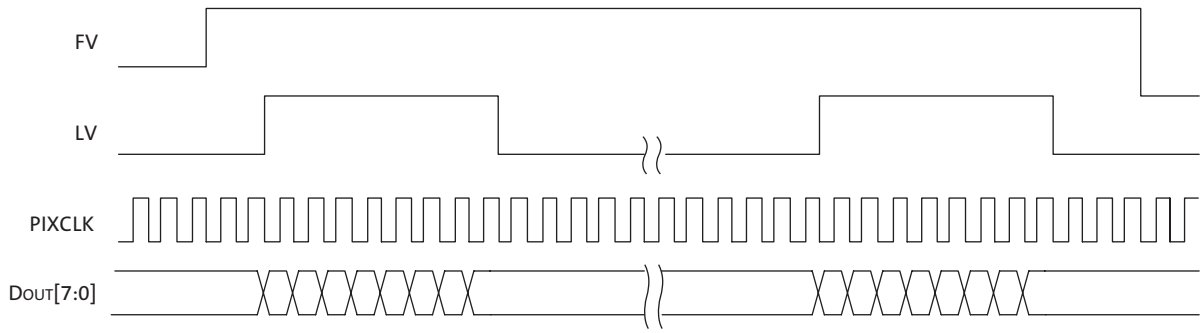
The following scenarios apply to bypass streams on the parallel output interface.

Case 1: Parallel Bypass Output with Clock Enabled

Figure 21 shows typical signal timing when the following options are applied.

1. Set insert_ccir_code = 0.
2. Set en_clk_between_lines = 1.
3. Set en_clk_between_frames = 1.

Figure 21: Timing of Parallel Bypass Output with Clock Enabled



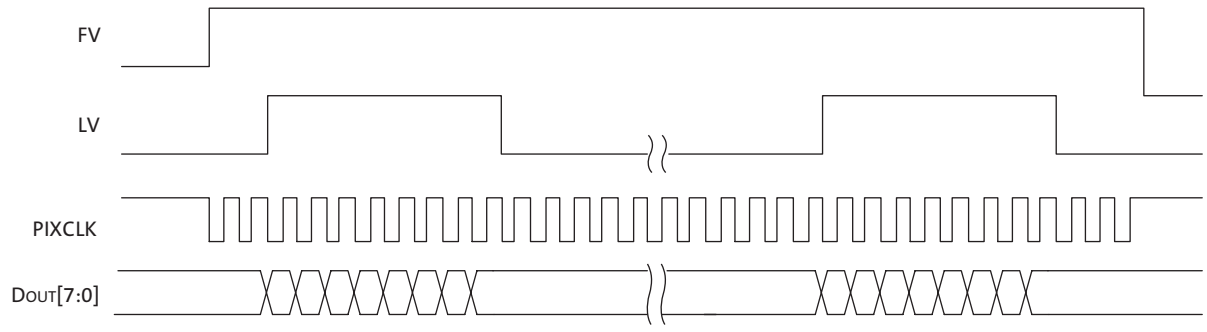
Notes: 1. Default PIXCLK is used in this example.

Case 2: Parallel Bypass Output with Clock Disabled Between Frames

Figure 22 shows typical signal timing when the following options are applied.

1. Set insert_ccir_code = 0.
2. Set en_clk_between_lines = 1.
3. Set en_clk_between_frames = 0.

Figure 22: Timing of Parallel Bypass Output with Clock Disabled Between Frames



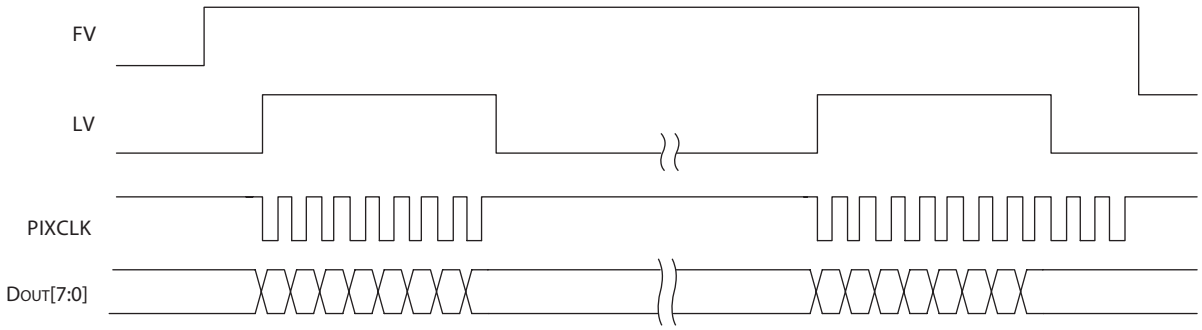
Notes: 1. Default PIXCLK is used in this example.

Case 3: Parallel Bypass Output with Clock Disabled Between Lines

Figure 23 shows typical signal timing when the following options are applied.

1. Set insert_ccir_code = 0.
2. Set en_clk_between_lines = 0.
3. Set en_clk_between_frames = 0.

Figure 23: Timing of Parallel Bypass Output with Clock Disabled Between Lines



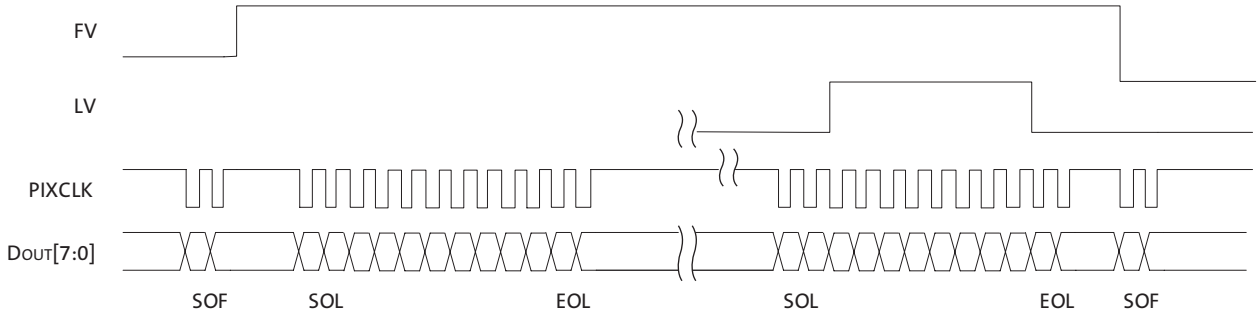
Notes: 1. Default PIXCLK is used in this example.

Case 4: Parallel Bypass Output with Clock Disabled and CCIR Codes Inserted

Figure 24 shows typical signal timing when the following options are applied.

1. Set insert_ccir_code = 1.
2. Set en_clk_between_lines = 0.
3. Set en_clk_between_frames = 0.

Figure 24: Timing of Parallel Bypass Output with Clock Disabled and CCIR Codes Inserted



Notes: 1. Default PIXCLK is used in this example.

JPEG Continuous Stream

The JPEG continuous stream only goes out through the parallel output interface, and supports the following features:

- Adaptive clock switching
- Insertion of SOI/EOI codes
- Duplication of FV on LV
- Appending JPEG status at the end of a data stream
- Insertion of thumbnail data

These features can be set in the output buffer TX control register. Thumbnail data will be embedded in the outgoing stream when programmed by the user.

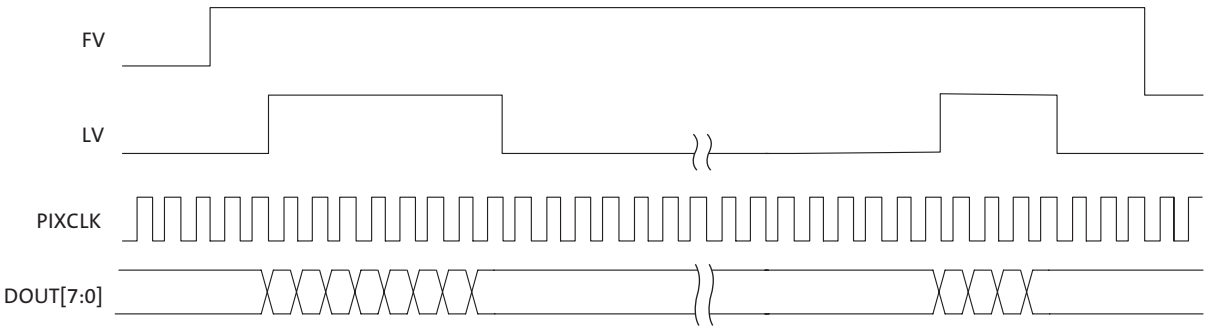
The following scenarios apply to continuous streams of image data.

Case 1: Parallel Output with Continuous Clock

Figure 25 shows typical signal timing when the following options are applied:

1. Set en_soi_eoi = 0.
2. Set en_clk_invalid_data = 1.
3. Set en_clk_between_frames = 1.
4. Set dup_fv_on_lv = 0.

Figure 25: Timing of Parallel Output with Continuous Clock



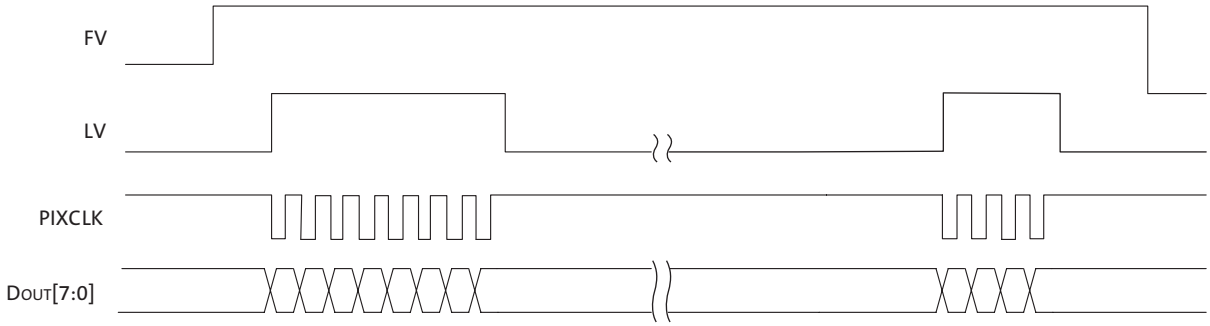
Notes: 1. Default PIXCLK is used in this example.

Case 2: Parallel Output with Gated Clock

Figure 26 shows typical signal timing when the following options are applied:

1. Set en_soi_eoi = 0.
2. Set en_clk_invalid_data = 0.
3. Set en_clk_between_frames = 0.
4. Set dup_fv_on_lv = 0.

Figure 26: Timing of Parallel Output with Gated Clock



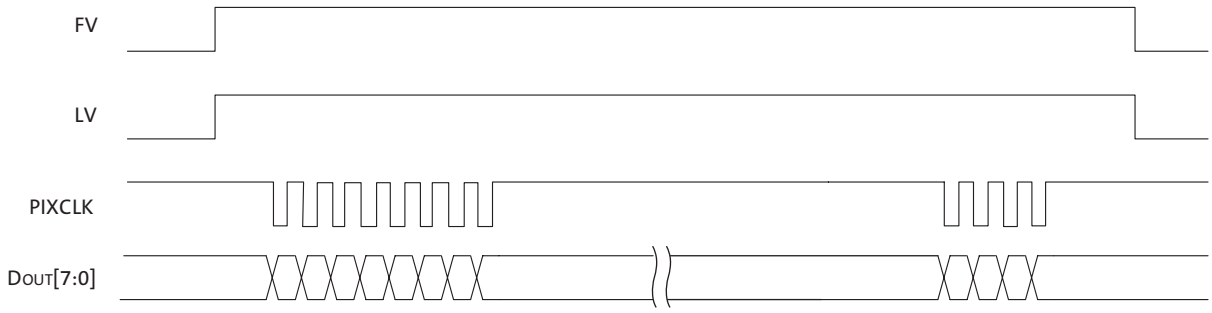
Notes: 1. Default PIXCLK is used in this example.

Case 3: Parallel Output When LINE_VALID is Enabled During FRAME_VALID

Figure 27 shows typical signal timing when the following options are applied:

1. Set en_soi_eoi = 0.
2. Set en_clk_invalid_data = 0.
3. Set en_clk_between_frames = 0.
4. Set dup_fv_on_lv = 1.

Figure 27: Timing of Parallel Output with Gated Clock



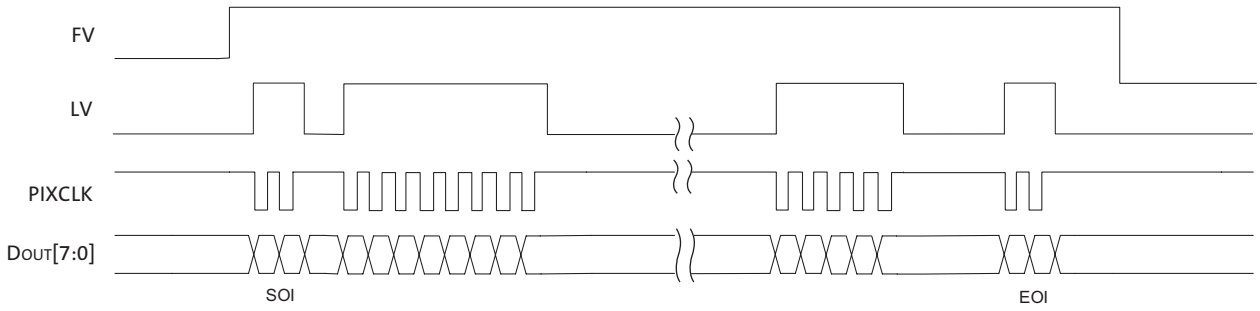
Notes: 1. Default PIXCLK is used in this example.

Case 4: Parallel Output When SOI and EOI Are Enabled During FRAME_VALID

Figure 28 shows typical signal timing when the following options are applied:

1. Set en_soi_eoi = 1.
2. Set soi_eoi_in_fv = 1.
3. Set en_clk_invalid_data = 0.
4. Set en_clk_between_frames = 0.

Figure 28: Timing of Parallel Output When SOI and EOI Are Enabled During FRAME_VALID



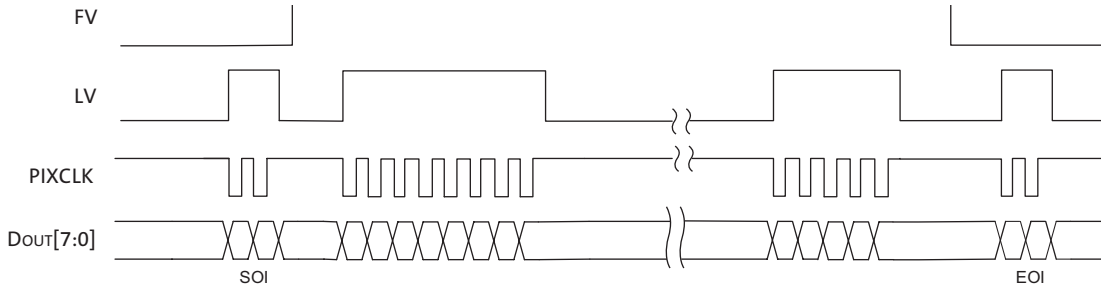
Notes: 1. Default PIXCLK is used in this example.

Case 5: Parallel Output When SOI and EOI Are Enabled But Not During FRAME_VALID

Figure 29 shows typical signal timing when the following options are applied:

1. Set en_soi_eoi = 1.
2. Set soi_eoi_in_fv = 0.
3. Set en_clk_invalid_data = 0.
4. Set en_clk_between_frames = 0.

Figure 29: Timing of Parallel Output When SOI and EOI Are Enabled But Not During FRAME_VALID



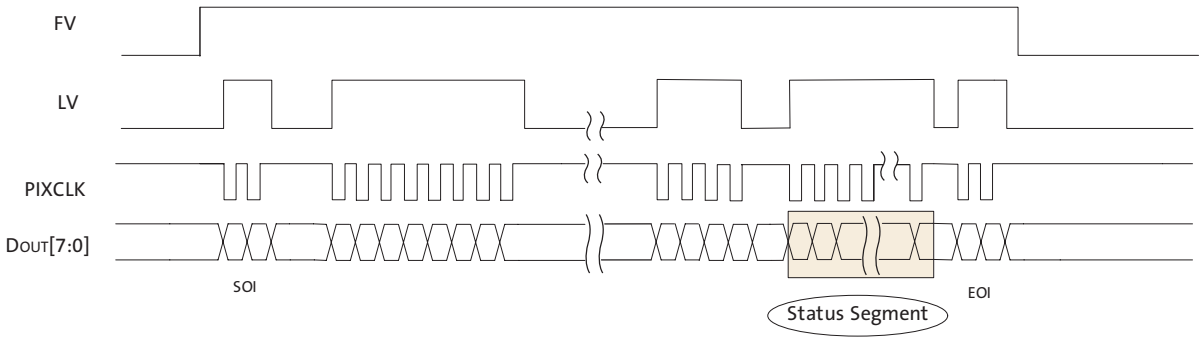
Notes: 1. Default PIXCLK is used in this example.

Case 6: Parallel Output with SOI/EOI, FRAME_VALID, and JPEG Status Inserted

Figure 30 shows typical signal timing when the following options are applied:

1. Set en_soi_eoi = 1.
2. Set soi_eoi_in_fv = 1.
3. Set insert_jpeg_status = 1.
4. Set en_clk_invalid_data = 0.
5. Set en_clk_between_frames = 0.

Figure 30: Timing of Parallel Output with SOI/EOI, FRAME_VALID, and JPEG Status Inserted



Notes: 1. Default PIXCLK is used in this example.

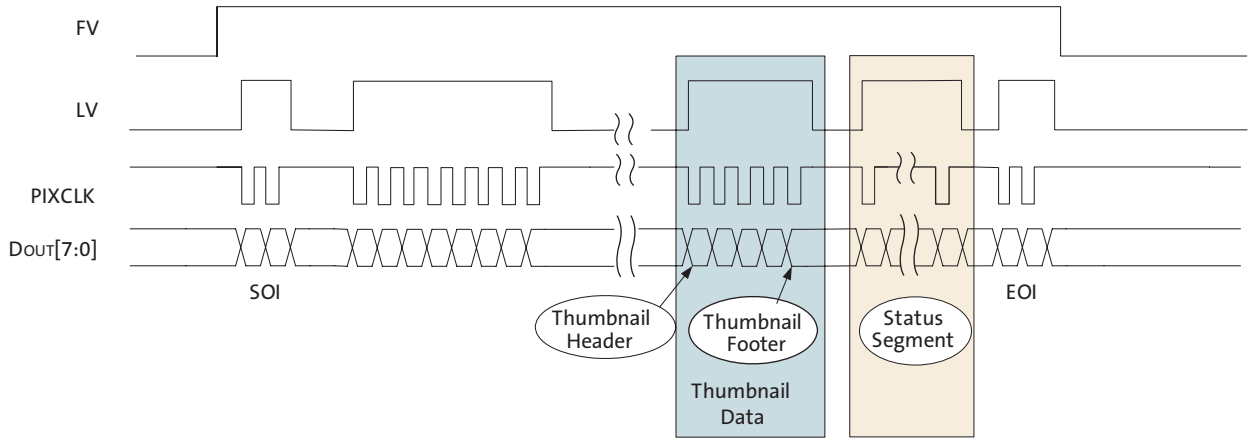
Case 7: Parallel Output with Embedded Thumbnail Data

Thumbnail data can be embedded in the JPEG continuous stream, which is delimited by the thumbnail header and footer.

Figure 31 shows typical signal timing when the following options are applied:

1. Set en_soi_eoi = 1.
2. Set soi_eoi_in_fv = 1.
3. Set insert_jpeg_status = 1.
4. Set en_clk_invalid_data = 0.
5. Set en_clk_between_frames = 0.

Figure 31: Timing of Parallel Output with Embedded Thumbnail Data



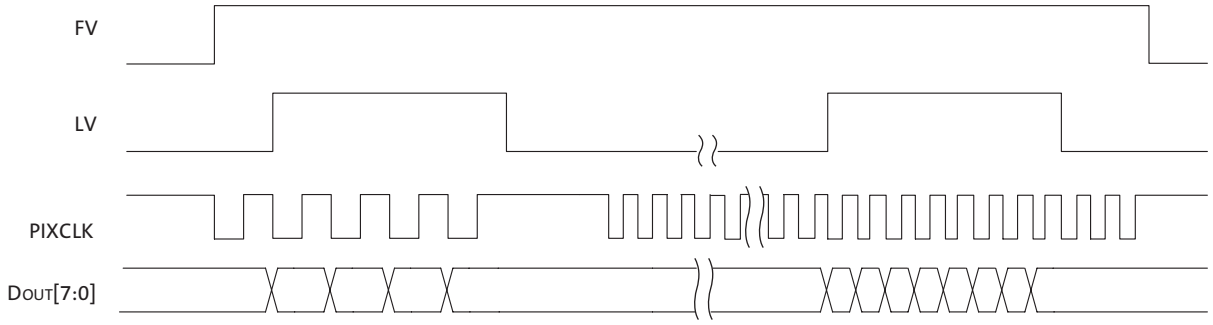
Notes: 1. Default PIXCLK is used in this example.

Case 8: Parallel Output with Adaptive Clock Switching

Adaptive clock switching is supported for a JPEG continuous stream. The switching can happen at any time during data transfer. Figure 32 shows typical signal timing when the following options and a continuous PIXCLK are applied:

1. Set en_soi_eoi = 0.
2. Set en_clk_invalid_data = 1.
3. Set en_clk_between_frames = 0.
4. Set en_adaptive_clk = 1.

Figure 32: Timing of Parallel Output with Adaptive Clock Switching



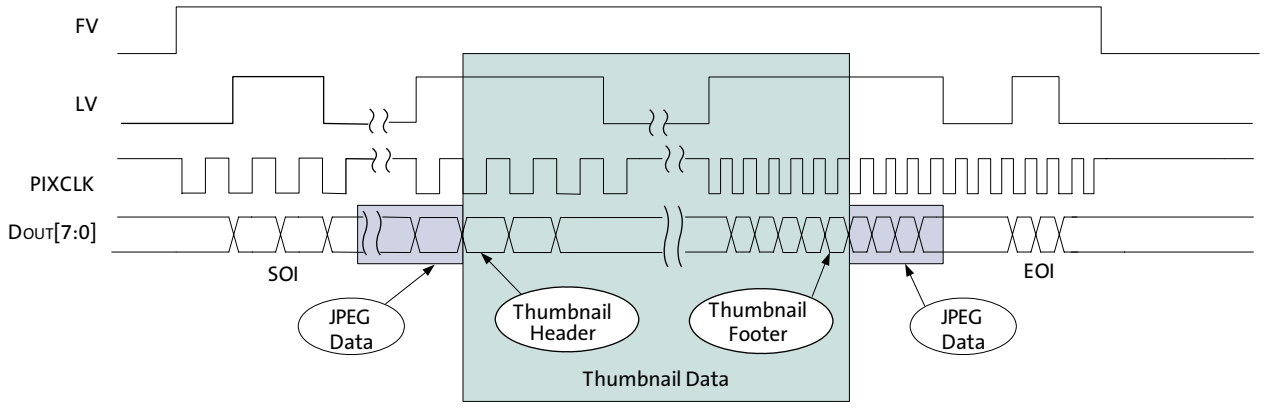
Notes: 1. Default PIXCLK is used in this example.

Case 9: Parallel Output with Adaptive Clock Switching and Embedded Thumbnail Data

Adaptive clock switching with thumbnail data is also supported for a JPEG continuous stream. The switching can happen at any time during data transfer. Figure 33 shows typical signal timing when the following options and a continuous PIXCLK are applied:

1. Set en_soi_eoi = 0.
2. Set en_clk_invalid_data = 1.
3. Set en_clk_between_frames = 0.
4. Set en_adaptive_clk = 1.

Figure 33: Timing of Parallel Output with Adaptive Clock Switching and Embedded Thumbnail Data



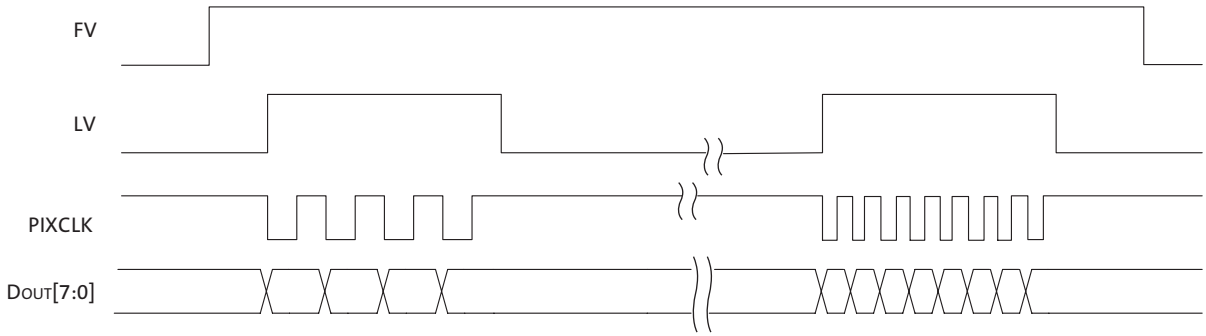
Notes: 1. Default PIXCLK is used in this example.

Case 10: Parallel Output with Gated PIXCLK

Figure 34 shows typical signal timing when the following options and a gated PIXCLK are applied:

1. Set en_soi_eoi = 0.
2. Set en_clk_invalid_data = 0.
3. Set en_clk_between_frames = 0.
4. Set en_adaptive_clk = 1.

Figure 34: Timing of Parallel Output with Gated PIXCLK



Notes: 1. Default PIXCLK is used in this example.

JPEG Spoof Stream

The JPEG compressed data can be output in spoof mode, which is defined in three registers—spoof_control, spoof_width, and spoof_height.

When register options en_soi_eoi and soi_eoi_in_fv are suppressed, it means there are no SOI or EOI in spoof mode.

Register option inser_ccir_code does not apply to JPEG spoof mode.

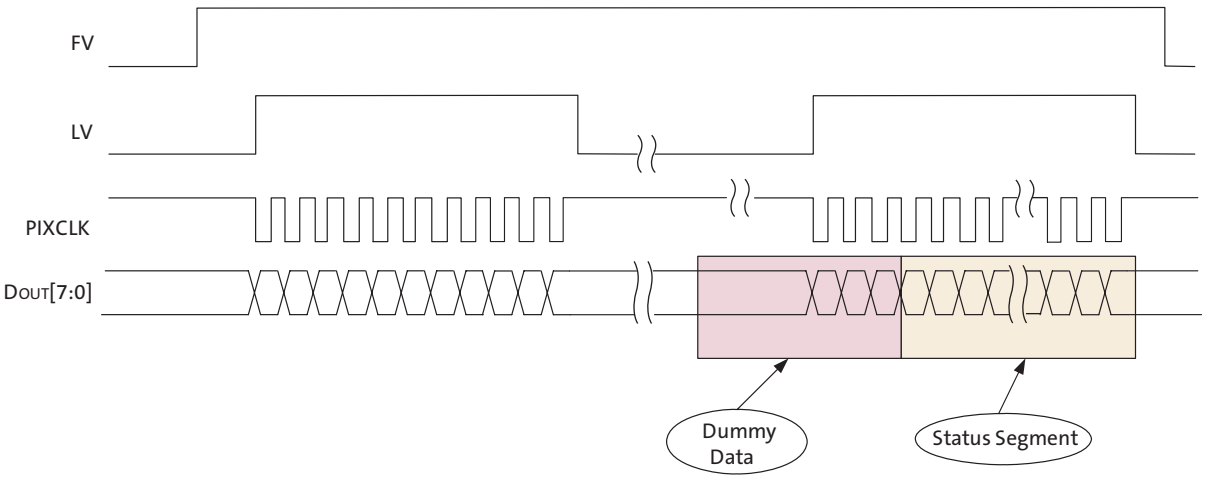
If the valid JPEG data is less than (*spoof_width* * *spoof height*), dummy data (“0”s) will be padded. It is mandatory that the last ten bytes of the stream are two bytes of SOSI code, four bytes of length information, two bytes of status, followed by two bytes of EOSI code. This means that the minimum spoof_width has to be greater than ten. Aptina recommends setting the minimum value of spoof_width to 16.

Case 1: PIXCLK Disabled Between Lines and Frames

Figure 35 shows typical timing when the following options are applied.

1. Set en_clk_between_lines = 0.
2. Set en_clk_between_frames = 0.

Figure 35: Timing of PIXCLK Disabled Between Lines and Frames



Notes: 1. Default PIXCLK is used in this example.

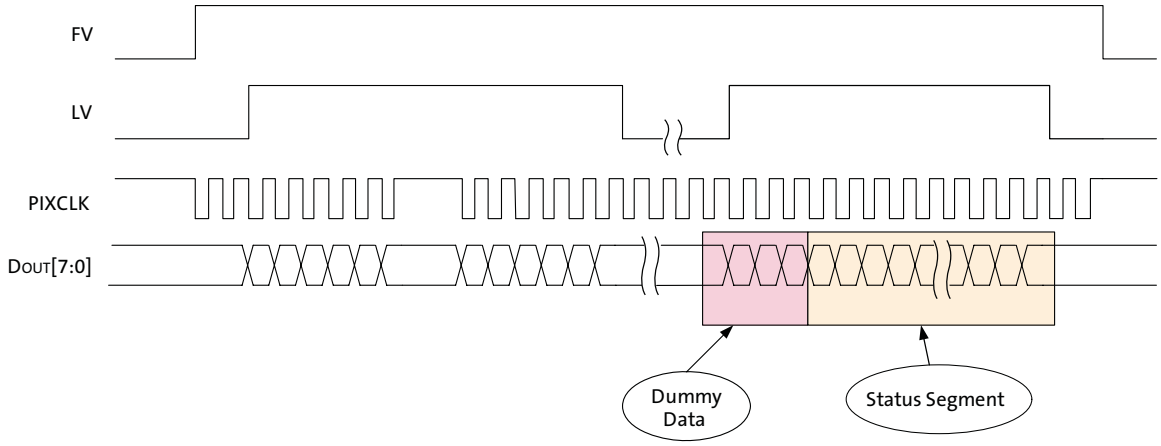
Case 2: PIXCLK Enabled Between Lines But Disabled Between Frames

If the data is not continuous, PCLK will be gated off to wait for valid data.

Figure 36 shoes typical timing when the following options are applied:

1. Set en_clk_between_lines = 1.
2. Set en_clk_between_frames = 0.

Figure 36: Timing of PIXCLK Enabled Between Lines But Disabled Between Frames



Notes: 1. Default PIXCLK is used in this example.

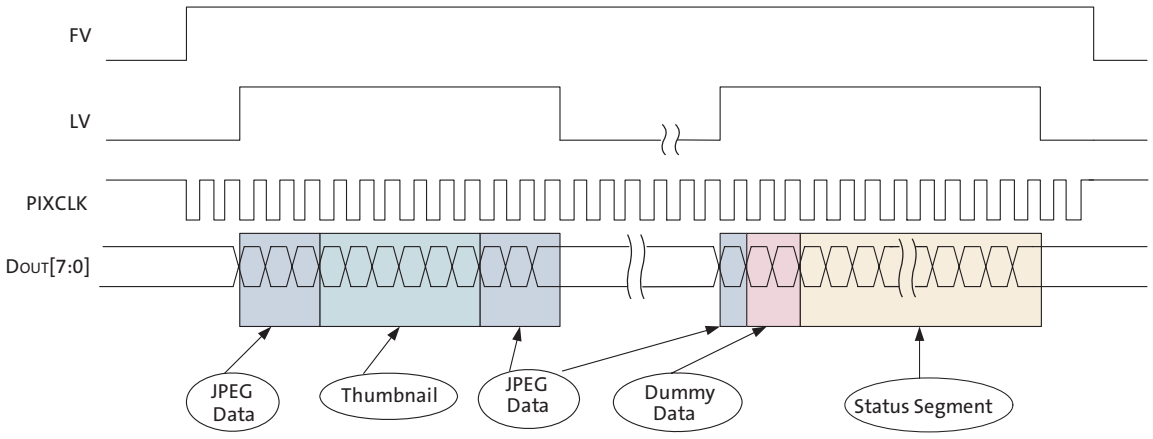
Case 3: Thumbnail Stream with One Frame of Data

When the thumbnail mode is enabled, the thumbnail stream may be inserted at any time and span more than one line.

Figure 37 shows typical timing when there is enough data to fill in one line after the thumbnail data, and the following options are applied:

1. Set EN_CLK_BETWEEN_LINES = 1.
2. Set EN_CLK_BETWEEN_FRAMES = 0.

Figure 37: Timing of Thumbnail Stream with One Frame of Data



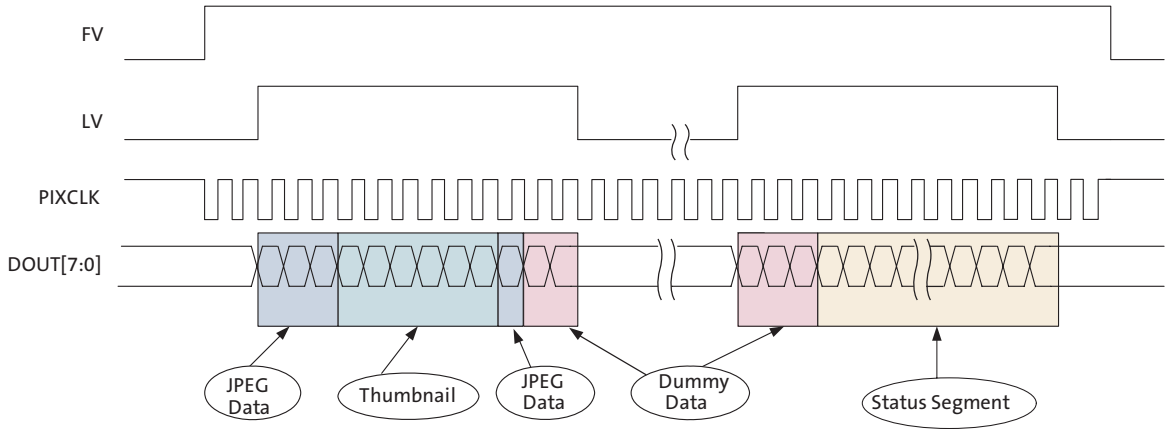
Notes: 1. Default PIXCLK is used in this example.

Case 4: Thumbnail Enabled with Less Than One Frame of Data

When the thumbnail mode is enabled, the thumbnail stream may be inserted at any time, and can span more than one line. Figure 38 shows typical timing when there is not enough data to fill in one line after the thumbnail data, and the following options are applied:

1. Set EN_CLK_BETWEEN_LINES = 1.
2. Set EN_CLK_BETWEEN_FRAMES = 0.

Figure 38: Timing of Thumbnail Stream with Less Than One Frame of Data



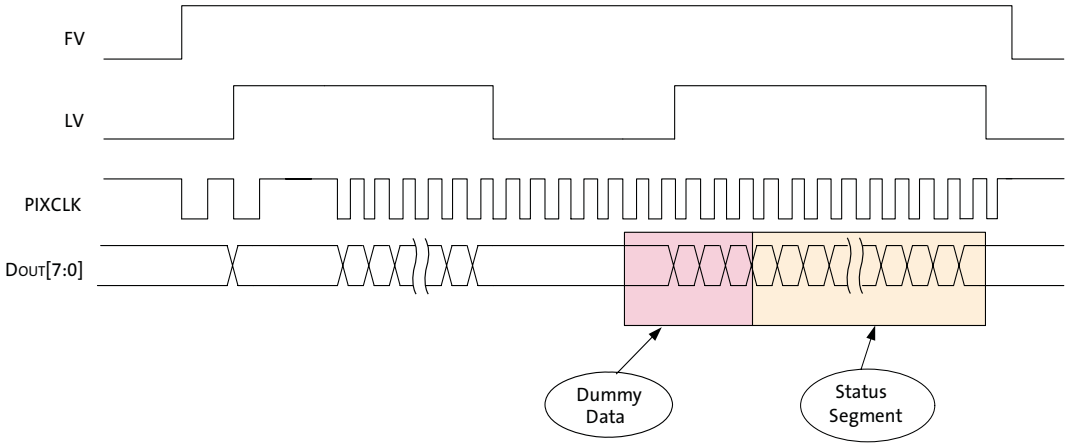
Notes: 1. Default PIXCLK is used in this example.

Case 5: Adaptive Clock Switching with PIXCLK Enabled Between Lines

Adaptive clock switching is supported for a JPEG spoof stream. The switching can happen at any time during data transfer. Figure 39 shows typical timing when the following options are applied:

1. Set en_clk_between_lines = 1.
2. Set en_clk_between_frames = 0.
3. Set en_adaptive_clk = 1.

Figure 39: Timing of Adaptive Clock Switching with PIXCLK Enabled Between Line



Notes: 1. Default PIXCLK is used in this example.

Output Data Format

The following output formats are available for the MT9T111.

- 16-bit YUV (default)
- 10-bit raw Bayer output
- 8-bit processed Bayer output
- (8 + 2)-bit Bayer output
- 16-bit 565RGB
- 15-bit 555RGB
- 12-bit 444xRGB

Since the data output signal is 8 bits wide, 12-, 15-, 16-, or (8+2)-bit data are output in two-byte sequence. For raw 10-bit Bayer data, two lowest significant bits are output on GPIO[1:0] signals.

Selecting Output Data Formats

Most of the variables used for context switching are located in the Cam_Pri_Context A (ID = 26), Cam_Pri_Context B (ID = 27), Cam_Sec_Context A (ID = 28), and Cam_Sec_Context B (ID = 29) variable map.

The MT9T111 can output several different formats—YCbCr, 565RGB, 555RGB, 444RGB, and raw data from offset 0x0007, as shown in Table 22.

Table 22: Changing Output Format Variables

0x0007 Bit Field	Output
9	Monochrome
8	Processed Bayer
7	Raw12-bit
6	Raw 10-bit
5	Raw 8-bit
4	12-bit 444xRGB
3	15-bit 555xRGB
2	16-bit 565RGB
1	YUV 4:2:0
0	YUV 4:2:2

A refresh command is needed (sequencer variable 0x00 = 0x6) before the new settings will be effective.

Table 23 shows other bits related to output format control from variable offset 0x0009.

Table 23: Output Format Option Configuration Settings

0x0009 Bit Field	Option/Configuration
8	Bayer output with first color to be Gb
7	Bayer output with first color to be B
6	Bayer output with first color to be R
5	Bayer output with first color to be Gr
4	Sampling mode for YUV422 with even U and odd V
3	Sampling mode for YUV422 with odd UV
2	Sampling mode for YUV422 with even UV
1	Swap the output high byte with low byte (swap byte order)
0	Swap red/blue or Cr/Cb channels (swap read and blue)

To select the 10-bit raw data from the sensor core, refer to the next section, "Outputting Raw Bayer Data".

Outputting Raw Bayer Data

There are two ways to obtain raw Bayer data. In both cases, the data from the sensor core will bypass the color pipeline.

The first option is to output all 10 bits in parallel. In this case, DOUT[7:0] represents sensor core data DOUT[9:2] and GPIO[1:0] represents sensor core data Dout[1:0].

With the color pipeline and MCU disabled, the sensor core parameters (integration time, gains, image size, power mode, and so on) can be programmed manually.

The second option is to enable 8+2 bypass mode by using DOUT[7:0] only. In this mode, the data bits are sent out in two bytes: DOUT[9:2] in the first byte, and DOUT[1:0] (with "0"s padded in the more significant bit positions) for the second byte.

YUV Output

The MT9T111 supports swapping YCrCb mode, as illustrated in Table 24.

Table 24: YCrCb Output Data Ordering

Mode				
Default (no swap)	Cbi	Yi	Cri	Yi+1
Swapped CrCb	Cri	Yi	Cbi	Yi+1
Swapped YC	Yi	Cbi	Yi+1	Cri
Swapped CrCb, YC	Yi	Cri	Yi+1	Cbi

RGB Output

The RGB output data ordering in default mode is shown in Table 25. The odd and even bytes are swapped when luma and chroma swap is enabled. R and B channels are bit-wise swapped when chroma swap is enabled.

Table 25: RGB Ordering in Default Mode

Mode (swap disabled)	Byte	D7 D6 D5 D4 D3 D2 D1 D0
565RGB	Odd	R7 R6 R5 R4 R3 G7 G6 G5
	Even	G4 G3 G2 B7 B6 B 5B4 B3
555RGB	Odd	0 R7 R6 R5 R4 R3 G7 G6
	Even	G4 G3 G2 B7 B6 B5 B4 B3
444xRGB	Odd	R7 R6 R5 R4 G7 G6 G5 G4
	Even	B7 B6 B5 B4 0 0 0 0
x444RGB	Odd	0 0 0 0 R7 R6 R5 R4
	Even	G7 G6 G5 G4 B7 B6 B5 B4

Walking 1s Test Pattern

The walking 1s test pattern meets the following requirements and conditions:

- The walking 1s test pattern will work in both preview and capture context (context A and context B)
- The MT9T111 must run in high power mode with no gated pixel clock
- The part must be put into the SOC bypass mode
  The output data is in RAW Bayer format
- The output resolution should be configurable
- The 8-bit pattern will use the same data bits as the YUV output
- The walking 1s test pattern is output only during active lines and frames, it is not output during blanking period
  The sequence is reset to start from "0" again, after the blanking
- The walking 1s test pattern is only available in parallel mode and not in MIPI mode

Figure 40: Sample Operation on One Line

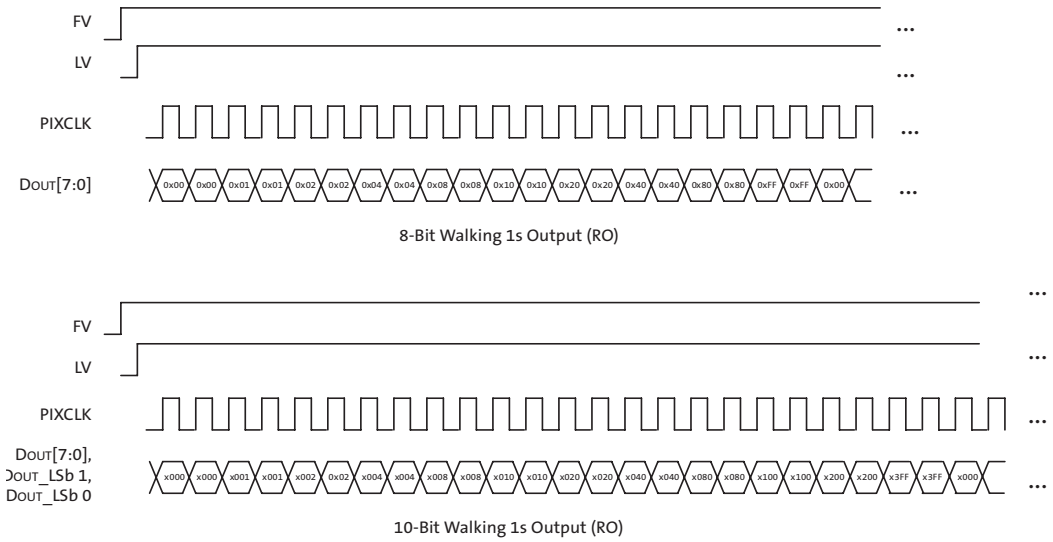
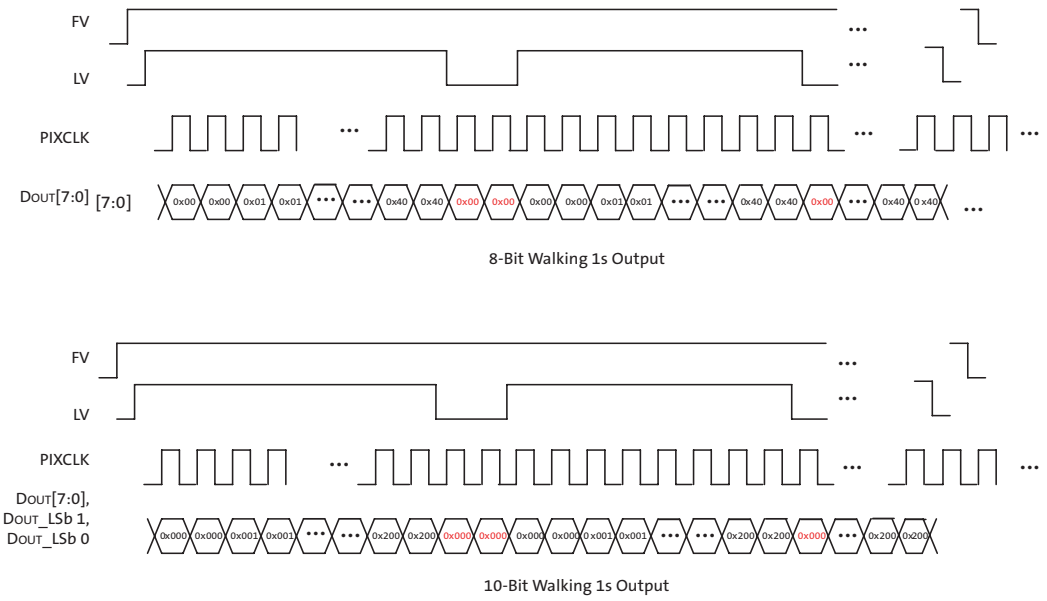


Figure 41: Sample Operation for Multiple Lines with Horizontal Blanking



Procedure

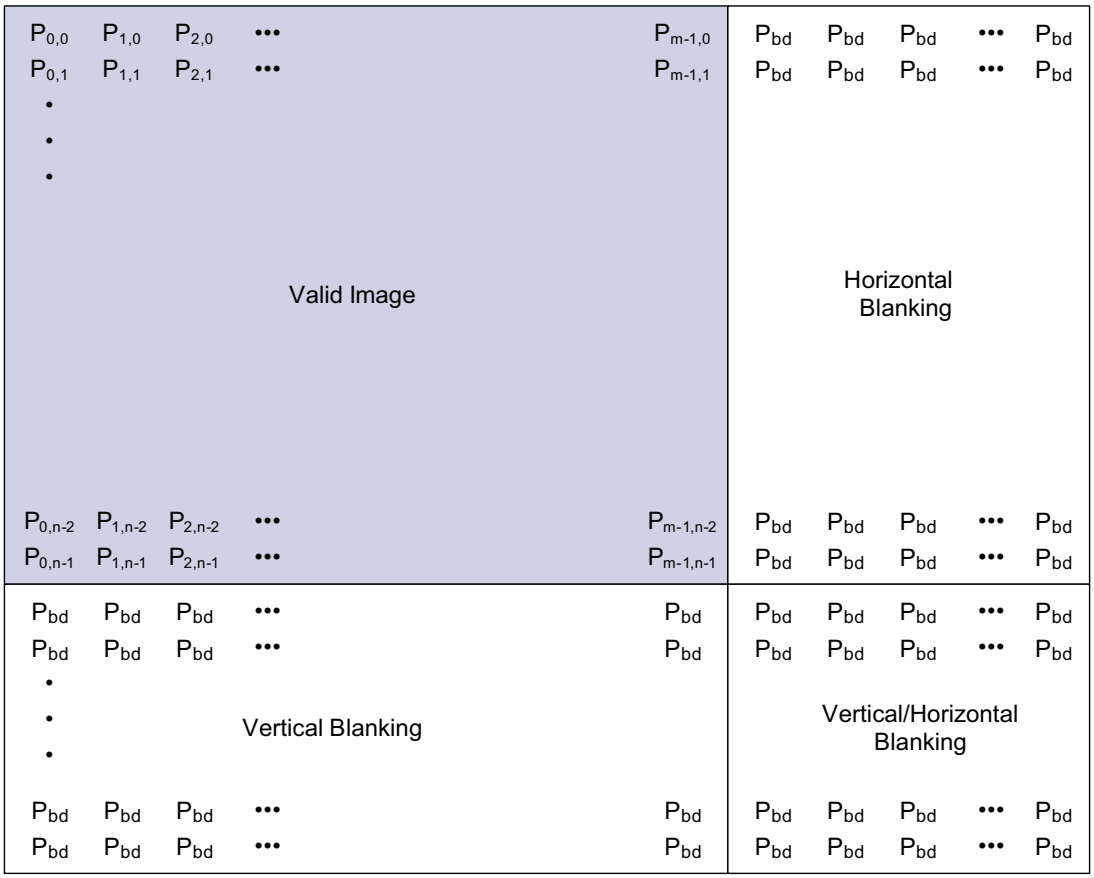
To enter the walking 1s test pattern:

1. Perform the power-on initialization routine.
2. Select the walking 1s test pattern data width by programming register map TX_SS 0x3220[1] to “1” for 10-bit data output and “0” for 8-bit data output mode.
3. Enable the walking 1s test pattern by setting SOC1 0x3290[5:4] to “10.”
4. Select the walking 1s test pattern output from the output multiplexer by programming SOC1 0x321C[3:0] = 0x4.

Sensor Core Interface

The sensor core image data is read out in a progressive scan. Valid image data is surrounded by horizontal blanking and vertical blanking, as shown in Figure 42. LV is HIGH during the shaded (Valid Image) region of the figure.

Figure 42: Spatial Illustration of Image Readout



Notes:
1. Px,y corresponds to 16-bit pixel output data at column = x, row = y with output window size of width = m, height = n. Last data output will be maintained during blanking time.
2. Data is output from:
   - 4a. Top left corner (0,0) across to (m-1, 0),
   - 4b. Horizontal blanking,
   - 4c. Next row (0,1) across to (m-1, 1),
   - 4d. Horizontal blanking,
   - 4e. Repeat rows until last valid image row (n-1),
   - 4f. Vertical blanking rows.
3. P_{bd} is indeterminate data during blanking time.

Mirroring and Flipping the Image

Mirroring and flipping of the image are operations of the MT9T111 sensor that can be configured in both contexts A and B independently. To perform mirroring and flipping, the user can initiate the operation with the following commands:

- Context A: Control driver ID 18, offset 0x0C, [1:0]
- Context B: Control driver ID 18, offset 0x54, [1:0]

The bit definitions follow:

- 0: Normal
- 1: Mirror horizontal
- 2: Flip vertical
- 3: Mirror and flip

The example below shows how to mirror an image on the horizontal axis:

```
VAR = 18, 0x0C, 0x146D  // READ MODE Register
VAR8 = 1, 0x00, 0x06  //  Refresh Command
```

The following commands show how to perform the same operation by using only register write commands:

```
REG = 0x098E, 0x480C
//Set the logical address to 0x480C (Page 18, 2 Byte Register, Offset 0x0C)
REG = 0x0990, 0x146D
// Set the given address to the value 0x146D
REG = 0x098E, 0x8400
// Set the logical address to 0x8400 (Page 1, 1 Byte Register, Offset 0x00)
REG = 0x0990, 0x06
// Set the given address to the value 0x06
```

Image Test Pattern Generation

The image test pattern generation block is located at the input section of the IFP block. The user can select an image test pattern output instead of the sensor output to test the full color pipeline process.

To enable test pattern generation, set driver ID 24, offset 0x03 = 0x100.

VAR = 24, 0x03, 0x0100

To select the color bar test pattern, modify driver ID 24, offset 0x25 = 0x06.

VAR = 24, 0x25, 0x06

The MT9T111 can generate the following test patterns:

VAR = 24, 0x25, 0x00: Test pattern disable
VAR = 24, 0x25, 0x01: Solid white test pattern
VAR = 24, 0x25, 0x04: Grey test ramp
VAR = 24, 0x25, 0x06: Color bar test pattern
VAR = 24, 0x25, 0x0A: Pseudo-random test pattern

To disable test pattern generation, set driver ID 24, offset 0x25 = 0x0.

VAR = 24, 0x25, 0x00

Then, set driver ID 24, offset 0x03 = 0x0.

VAR = 24, 0x03, 0x0000

Actual image test patterns depend on algorithms when the MCU is running. Therefore, the IFP will apply AWB to the image test pattern, process lens-shading correction, apply tonal and gamma curves, and modify digital gains.

Programming and Operation

The host processor programs registers and variables through the two-wire serial interface. The two-wire serial interface always operates in 16-bit address and 16-bit data transfer mode. Most registers (except GPIO registers) can be addressed directly by the host processor. For example, the host processor can send 16-bit address and data of corresponding registers directly over the two-wire serial interface.

For variables and GPIO registers, the host processor can access these through the XDMA registers. The XDMA register provides an address register (0x098E), and an array of data registers (0x0990–0x099E). The host processor must write a variable address value to the address register then write variable data values to data register. The XDMA block will translate both address and data to the corresponding memory locations. The details of the physical interfaces are shown in Figure 43.

Figure 43: Register and Variable Interfaces

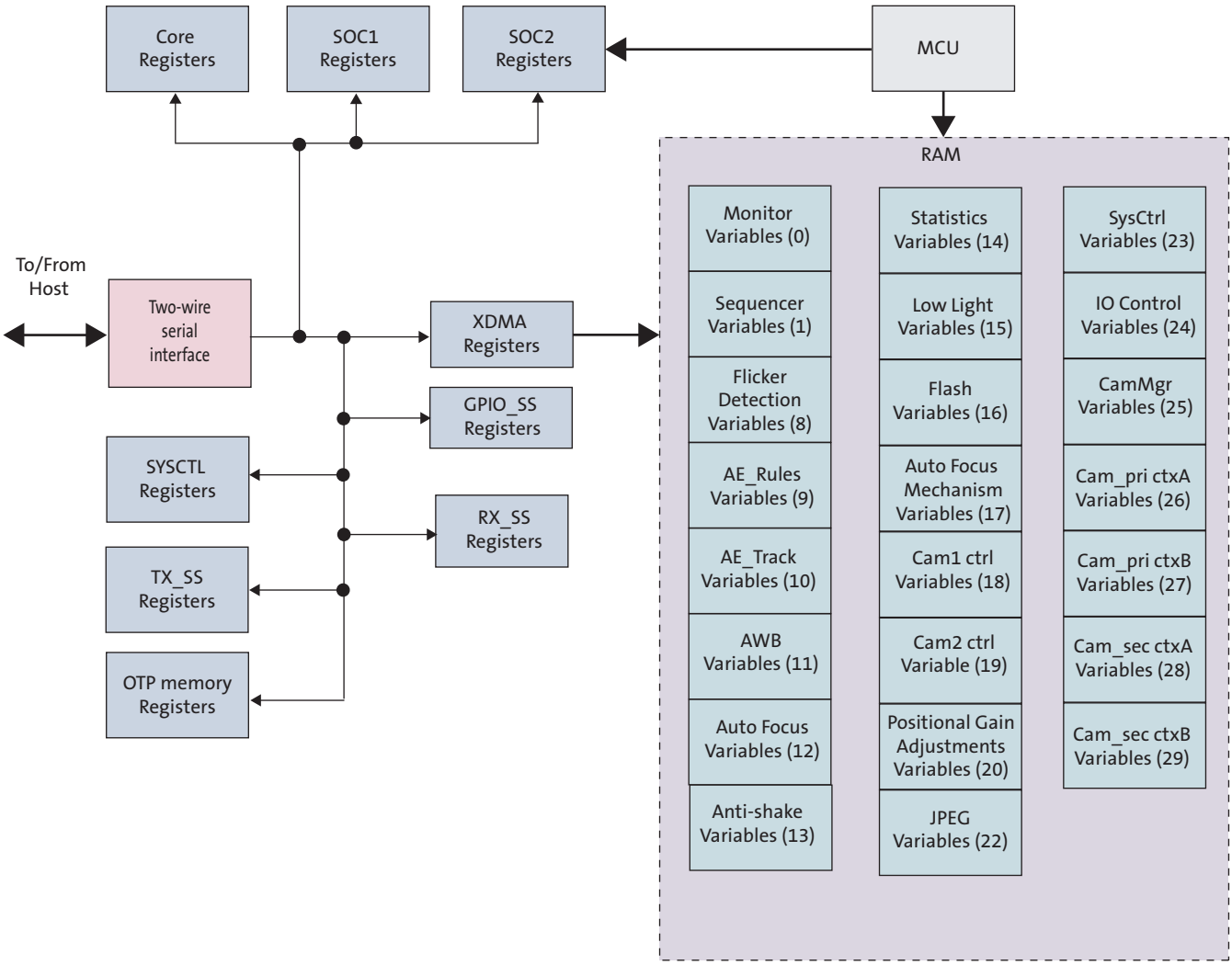


Table 26: Summary of MT9T111 Registers and Variables

Registers		Address Range		0x098A	Map Name	Descriptions
		0x0600–0x0614			GPIO_SS	GPIO, VGPIO, external sensor control
		0x3000–0x31FE			Core	Sensor control
		0x3800–0x3802			OTPM	OTP memory programming register
		0x3210–0x33FC			SOC1	Color pipeline/output format
		0x3400–0x3B86			SOC2	MIPI/PGA
		0x0000–0x0050			SYSCTL	Clock/reset/standby control
		0x0100–0x0158			RX_SS	RX FIFO and test pattern control
		0x0982–0x099E			XDMA	Indirect access to MCU memory
		0x3C00–0x3CEC			TX_SS	Output control/JPEG
Variables : ID DEC	**HEX**	**Address Offset**	**0x098E Start Address 8-bit Register**	**0x098E Start Address 16-bit Register**	**Map Name**	**Descriptions**
0	00	0x0000–0x0027	0x8000	0x0000	Monitor	Patch control
1	01	0x0000–0x0004	0x8400	0x0400	Sequencer	Refresh command
8	08	0x0000–0x0017	0xA000	0x2000	FD (Flicker detection)	Flicker control
9	09	0x0000–0x0037	0x400	0x2400	AE_Rule	AE control
10	0A	0x0000–0x0030	0xA800	0x2800	AE_Track	AE control
11	0B	0x0000–0x0049	0xAC00	0x2C00	AWB	AWB control
12	0C	0x0000–0x0046	0xB000	0x3000	AF (Auto Focus)	Auto focus control
13	0D	0x0000–0x003C	0xB400	0x3400	Anti-Shake	Anti-shake control
14	0E	0x0000–0x0084	0xB800	0x3800	Stat (Statistics)	
15	0F	0x0000–0x0054	0xBC00	0x3C00	Low light	Gamma curve
16	10	0x0000–0x0027	0xC000	0x4000	Flash	Flash control
17	11	0x0000–0x0034	0xC400	0x4400	AFM	Auto focus mechanism
18	12	0x0000–0x014A	0xC800	0x4800	CAM1 control	Camera 1 (internal sensor) control, CCM
19	13	0x0000–0x014A	0xCC00	0x4C00	CAM2 control	Camera 2 (external sensor) control, CCM
20	14	0x0000–0x000A	0xD000	0x5000	PGA	PGA status
22	16	0x0000–0x0009	0xD800	0x5800	JPEG	JPEG status
23	17	0x0000–0x003E	0xDC00	0x5C00	SysCtrl	SysCtrl status
24	18	0x0000–0x002E	0xE000	0x6000	IO control	Image test pattern
25	19	0x0000–0x0007	0xE400	0x6400	CamMgr	Camera status
26	1A	0x0000–0x00AC	0xE800	0x6800	Cam_Pri context A	Camera 1 context A settings
27	1B	0x0000–0x00AC	0xEC00	0x6C00	Cam_Pri context B	Camera 1 context B settings
28	1C	0x0000–0x00AC	0xF000	0x7000	Cam_Sec context A	Camera 2 context A settings
29	1D	0x0000–0x00AC	0xF400	0x7400	Cam_Sec context B	Camera 2 context B settings

Power-On Operation

The following list is an example of the host processor commands to be sent to the MT9T111 after the power-on sequence. Depending on the application, the command sequence can be changed or omitted.

1. The system powers up. All the power supplies are turned on and the external clock (EXTCLK) is running. RESET and STANDBY signals are in an inactive state.
2. The PLL is disabled and in bypass mode.
3. Program the PLL and clock dividers.
4. Enable the PLL.
5. Wait for the PLL to lock.
6. Turn off the PLL bypass mode.
7. Set the output slew rate.
8. Enable parallel output.
9. Remove sensor from soft standby.
10. Load PGA coefficients for LSC settings.
11. Load CCM table values for AWB settings.

Example of ini File for Power-On Sequence:

The following examples show more detail for the power-on operation steps shown above.

```
[ Basic Initialization]
// Reset Sensor
REG = 0x001A, 0x0219 // RESET_AND_MISC_CONTROL
DELAY=1
REG= 0x001A, 0x0218 // RESET_AND_MISC_CONTROL
// wait for Sensor to come out of reset
POLL_REG = 0x0000,0xFFFF, ! = 0x2680, DELAY = 10,TIMEOUT = 100
//DELAY=300

// Setup PLL
|LOAD = PLL Settings

// Enable Clock Pad
REG =  0x0016, 0x0400 // CLOCKS_CONTROL
// Enable Parallel Output
REG = 0x001A, 0x0218 // RESET_AND_MISC_CONTROL
// Maximize Slew Rate
REG = 0x001E, 0x0777 // PAD_SLEW_PAD_CONFIG
// Remove Sensor from Standby
REG = 0x0018, 0x4028 // STANDBY_CONTROL_AND_STATUS
// Wait for Sensor to come out of Standby
POLL_REG=0x0018, 0x4000, ! = 0x0000, DELAY = 10, TIMEOUT = 100
//DELAY = 300
```

Standby

The MT9T111 may be placed in standby mode either through hardware—by externally driving the STANDBY signal—or through software—by writing to a register through the two-wire serial interface. The internal operating mode is similar when the device enters standby through hardware or software. However, in different standby modes, the device has different power consumption. In soft standby mode, the two-wire serial interface is active, EXTCLK is active, all registers are saved, and all context information is saved. In hard standby mode, the two-wire serial interface is not active, all context information and PLL settings are saved, and EXTCLK is gated off. Under default conditions other registers are not retained.Register values can be retained in hard standby mode through programming.

Table 27: Standby Operation in Different Modes

Operation/Condition	Soft Standby	Hard Standby with Shutdown (Default)	Hard Standby with Memory Retention
Trigger to enter standby	Set standby_i2c bit (0x0018[0] = 1)	Assert STANDBY signal[1]	Assert STANDBY signal[1]
Trigger for leaving standby	Clear standby_i2c bit (0x0018[0] = 0)	De-assert STANDBY signal	De-assert STANDBY signal
Final State of standby	Parallel output = High-Z, MIPI output = 0	Parallel in High-Z and MIPI output = 0	Parallel in High-Z and MIPI output = 0
Clocks gated off	Internal clocks gated off, two-wire serial I/F active	Internal clocks gated off; two-wire serial I/F inactive	Internal clocks gated off; two-wire serial I/F inactive
Register/SRAM contents	Register and memory state kept	Register and memory state are lost	Register and memory state kept
Power consumption	Internal power on, Clocks not gated off, Some power saved	Internal power down, Clocks gated off, Lowest power state	Internal power on, Clocks gated off, 2nd lowest power state
State after resume from standby	Resume to state before standby	Previous state lost, POR issued	Resume to state before standby

Notes: 1. The sensor must complete the basic initialization sequence described on "Power-On Operation" on page 77 after power up, before asserting the STANDBY signal.

Entering Standby Mode

Soft standby can be entered using following commands from the host to the MT9T111:

1. Set 0x0028[0] (en_vdd_dis_soft bit) = 0.
2. Set 0x0018[0] (standby_i2c bit) = 1.
3. Read 0x0018[14] (standby_done bit = 1) for standby status.

Hard standby with shutdown can be entered using following commands:

1. Set 0x0028[0] (en_vdd_dis_soft bit) = 1 (default state).
2. Assert STANDBY.

Hard standby with memory retention can be entered using following commands:

1. Set 0x0028[0] (en_vdd_dis_soft bit) = 0.
2. Assert STANDBY.

Exiting Standby Mode

To exit the soft standby mode:

1. The host system sets 0x0018[0] (standby_i2c bit) = 0.

To exit hard standby with shutdown:

1. De-assert STANDBY.
2. The host programs the start up settings for CCM, lens shading correction, and so forth.

Timing Specifications

Power-Up Sequence for Rev2 Silicon

Powering up the Rev2 sensor requires voltages to be applied in a particular order, as shown in Figure 44. The timing requirements are shown in Table 28. It is advised that the user manually assert a hard reset upon power up for Rev2.

Figure 44: Power-Up Sequence Rev2 silicon

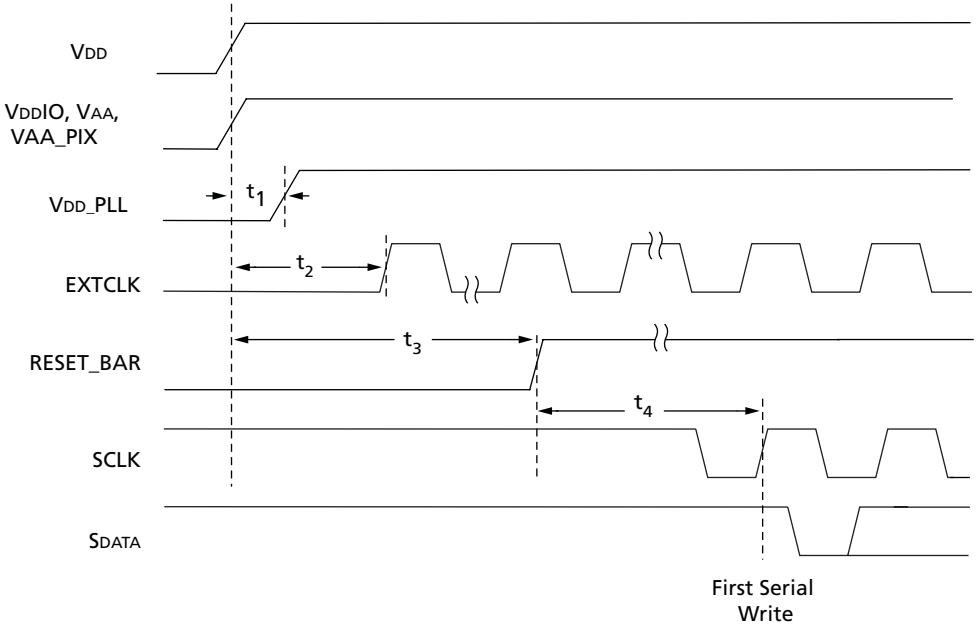


Table 28: Power-Up Signal Timing for Rev2 Silicon

Parameter	Symbol	Min	Typ	Max	Unit
VDD to VDDIO, VAA, and VAA_PIX	–	0	–	500	ms
VDD to VDD_PLL	t_1	1	–	500	
VDD to EXTCLK Activation	t_2	–	500	–	
RESET_BAR activation time	t_3	70	–	–	EXTCLKs
First serial write	t_4	100	–	–	EXTCLKs

Power-up Sequence for Rev3 Silicon

Powering up the Rev3 sensor is independent of voltages applied in a particular order, as shown in Figure 45. The timing requirements for other signals are shown in Table 29. It is advised that the user manually assert a hard reset upon power up for Rev3.

Figure 45: Power-Up Sequence Rev3 silicon

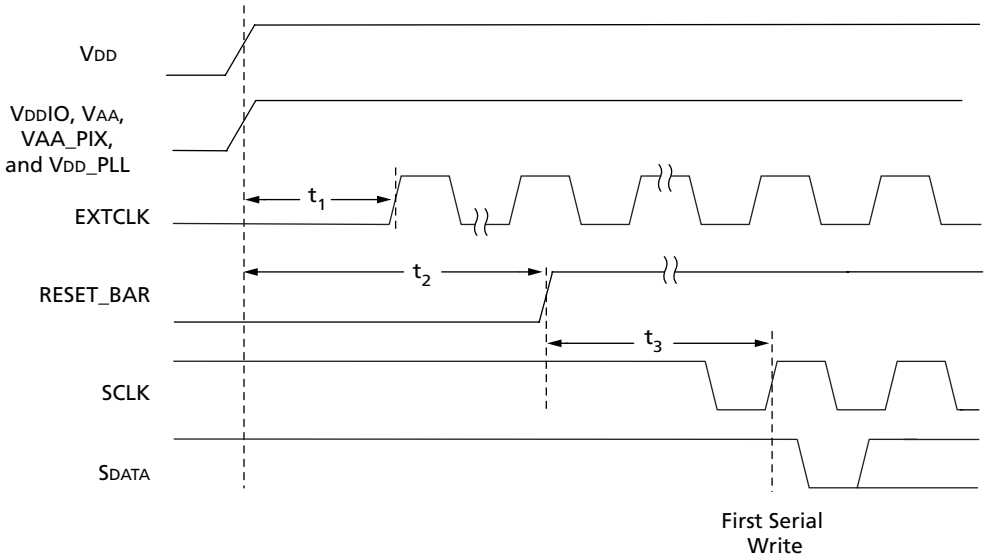


Table 29: Power-Up Signal Timing Rev3 Silicon

Parameter	Symbol	Min	Typ	Max	Unit
VDD to VDDIO, VAA, VDD_PLL, and VAA_PIX	–	–	–	500	ms
VDD to EXTCLK Activation	t1	–	500	–	
RESET_BAR activation time	t2	70	–	–	EXTCLKs
First serial write	t3	100	–	–	EXTCLKs

Reset

Two types of reset are available:

- A hard reset is issued by toggling RESET_BAR
- A soft reset is issues by writing commands through the two-wire serial interface

Hard Reset

After hard reset, the output FIFO is configured for operation, but disabled, and all outputs are tri-stated. These outputs can be enabled through the two-wire serial interface. The hard reset signal sequence is shown in Figure 46, and the hard reset timing is shown in Table 30.

Figure 46: Hard Reset Signal Sequence

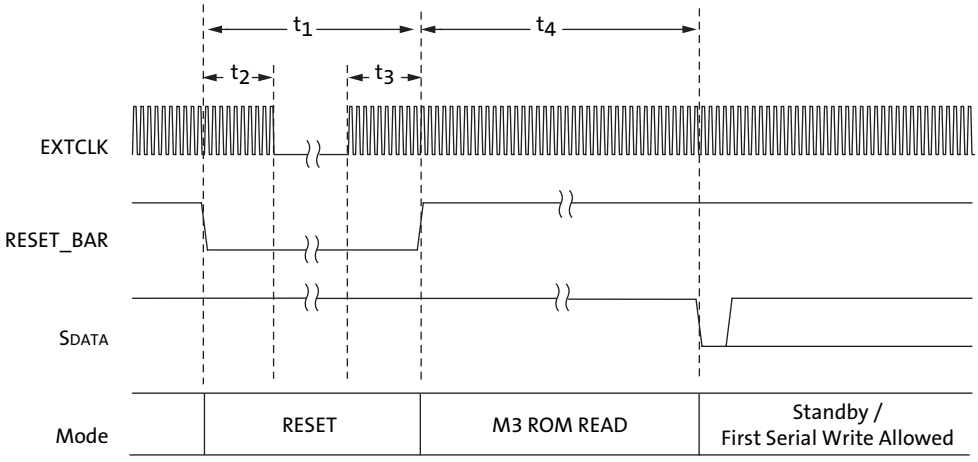


Table 30: Hard Reset Signal Timing

Parameter	Symbol	Min	Typ	Max	Unit
RESET_BAR pulse width	$^{t}1$	70	–	–	EXTCLKs
Active ECXTCLK after RESET_BAR is asserted	$^{t}2$	10	–	–	
Active EXTCLK before RESET_BAR is de-asserted	$^{t}3$	10	–	–	
First two-wire serial interface communication after RESET is HIGH	$^{t}4$	–	100	–	

Soft Reset

A soft reset sequence to the sensor has the same affect as the hard reset, and can be activated by writing to a register through the two-wire serial interface. The soft reset signal sequence is shown in Figure 47, and the soft reset timing is shown in Table 31.

Figure 47: Soft Reset Signal Sequence

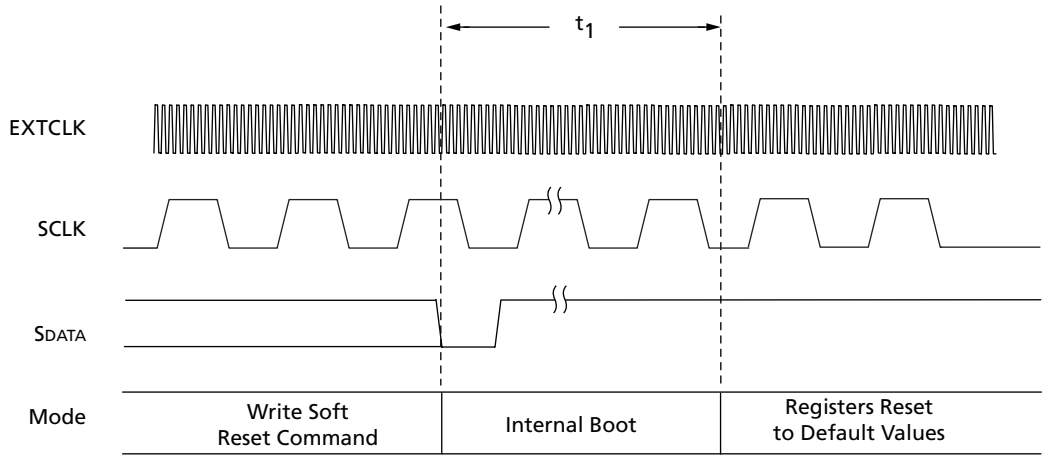


Table 31: Soft Reset Signal Timing

Parameter	Symbol	Min	Typ	Max	Unit
Active EXTCLK after soft reset command is asserted	t_1	–	100	–	EXTCLKs

Standby Modes

The MT9T111 supports three different standby modes:

- Hard standby with shutdown
- Hard standby with memory retention
- Soft standby with state retention

For all hard and soft standby modes, entry can be inhibited by programming the standby_control register.

Hard Standby with Shutdown Mode

The hard standby with shutdown mode uses STANDBY to shut down digital power (V_{DD}), and ensure the lowest power consumption. All the two-wire serial interface settings and firmware variables, including patches, will be lost in this mode. Starting up from this mode is similar to performing a power up. The host will not have to reload the PLL and clock divider settings, but other sensor settings such as sensor timing, LSC, CCM, and so forth, will have to be reloaded. The two-wire serial interface will be inactive and the sensor must be started up by de-asserting STANDBY.

Hard Standby With Memory Retention Mode

Hard standby with memory retention mode (without the loss of variable data) can also be achieved. This mode stores the variables and state of the sensor before entering standby (similar to soft standby). The power consumption is lower than that of soft standby, as internal clocks are turned off, and the two-wire serial interface is inactive.

Since the hard standby with memory retention mode is activated by STANDBY, the en_vdd_dis_soft register needs to be programmed to indicate the selection of this mode, before STANDBY is asserted. De-asserting STANDBY causes the sensor to come out of standby mode. This also causes the sensor to resume operation from the state before the STANDBY signal was asserted. By default, asserting the STANDBY signal causes the hard standby mode described above.

The signal sequence for both modes is shown in Figure 48, and the timing for both modes is shown in Table 32 on page 85.

By default, asserting the STANDBY signal causes the hard standby with shutdown mode (see Hard Standby with Shutdown Mode) above.

Figure 48: Hard Standby Signal Sequence Mode

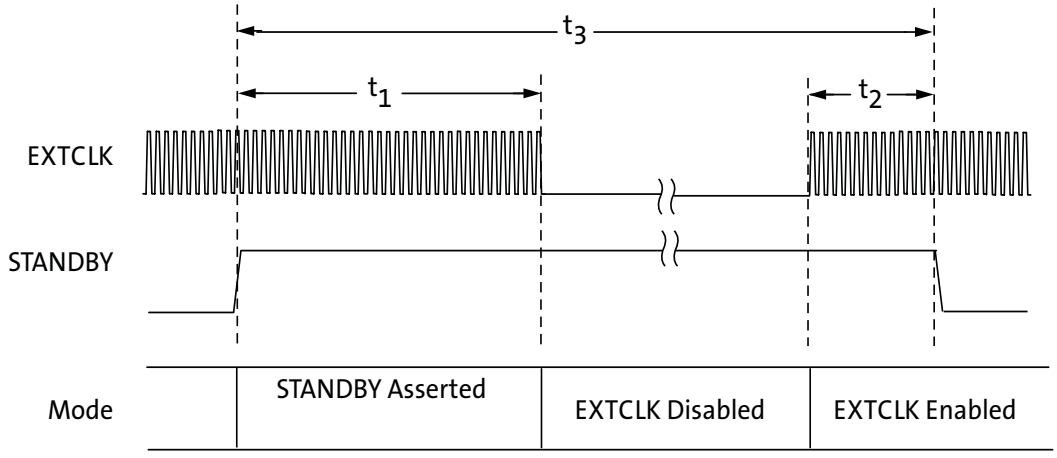


Table 32: Hard Standby Signal Timing

Parameter	Symbol	Min	Typ	Max	Unit
Standby entry complete	t_1	20	–	–	?s
Active EXTCLK before STANDBY de-asserted	t_2	10	–	–	
STANDBY pulse width	t_3	100	–	–	

Soft Standby with State Retention

Soft standby with state retention can be enabled by register access, disabling the sensor core and most of the digital logic. The two-wire serial interface is still active and the sensor can be programmed through register commands. All register settings and RAM content will be preserved. Soft standby can be performed in any sequencer state after all AE, AWB, histogram, and flicker calculations are finished, and the sensor core has been disabled.

The execution of standby will take place after the completion of the current line by default. It is possible to synchronize the execution of standby with the end of frame through the standby_control register. The soft standby signal sequence is shown in Figure 49, and the signal timing is shown in Table 33.

Figure 49: Soft Standby Signal Sequence

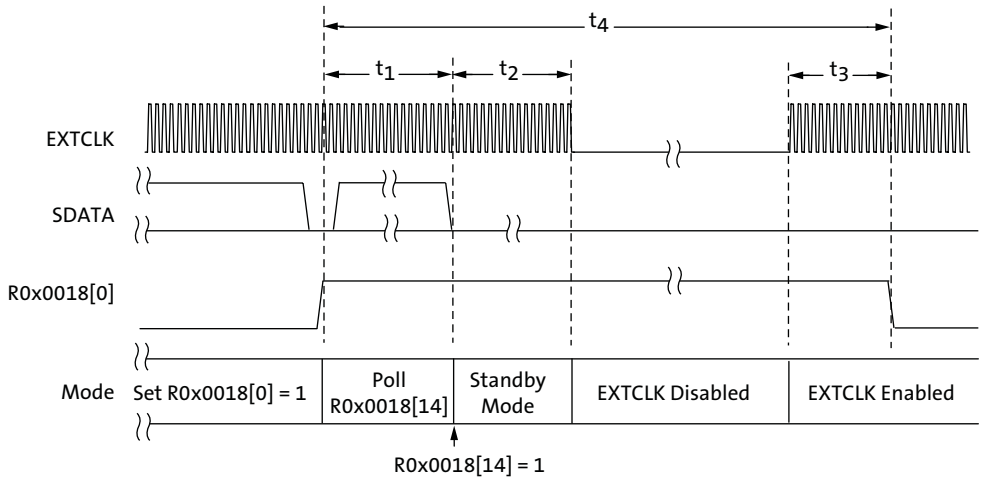


Table 33: Soft Standby Signal Timing

Parameter	Symbol	Min	Typ	Max	Unit
Standby entry complete	t_1	20	–	–	?s
Active EXTCLK before soft standby de-activates	t_2	10	–	–	
Minimum standby time	t_3	100	–	–	

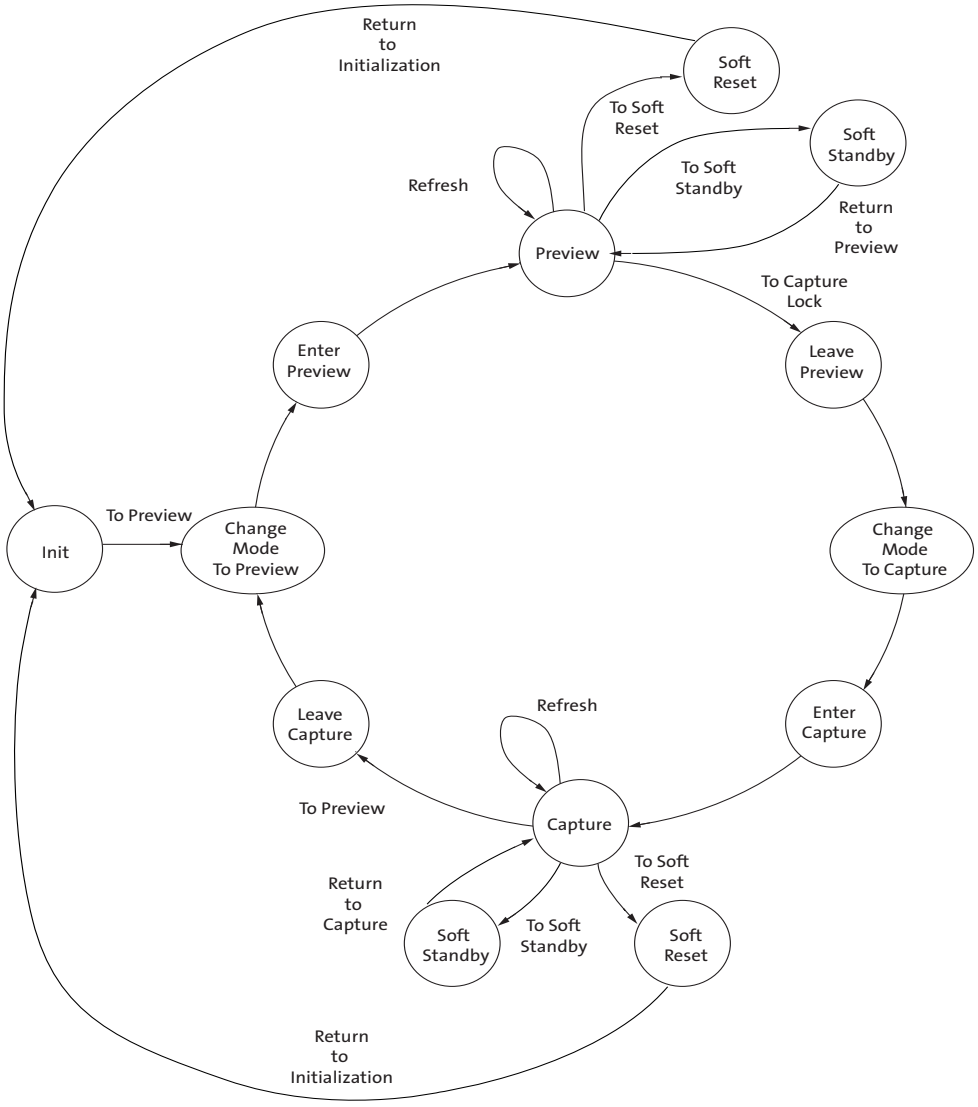
Image Signal Processing Flow and Camera Control

The advanced image flow processor (IFP) and flexible programmability of the MT9T111 provide a variety of ways to enhance and optimize the image sensor performance. Built-in optimization algorithms enable the MT9T111 to operate with factory settings as a fully automatic, highly adaptable camera; however, most of its settings are user-programmable.

Context Switching and Output Configuration

There are two contexts (or modes) available, A and B. Context A is known as the preview mode and has a default resolution of 1024 x 768, while context B is called the capture mode with a default resolution of 2048 x 1536. The context switching sequence is shown in Figure 50.

Figure 50: State Machine for Context Switching



There is a set of variables for each context that enables the user to configure its properties. These settings are automatically put into effect by the MCU during context switching. Examples of configurable options are: output resolution, crop sizes, data format, output FIFO, special effects, and gamma table.

Setting up Preview (A) and Capture (B) Modes

Mode variables store 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 mid-frame changes.

The mode variable 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 mid-frame 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 capture modes.

Upon power up, all non-default register values desired should be uploaded, including the mode variable values. Hardware registers that correspond to the mode variable values do not need to be uploaded because the registers are overwritten upon initialization, a context change (preview to capture or vice versa), or a sequencer REFRESH command.

These mode variables can be changed at any time, but their changes are not reflected in the images until one of the following occurs:

- A REFRESH or REFRESH_MODE command to the sequencer is issued.
- The state changes from preview to capture (state 3 to state 7) or vice versa (state 7 to state 3).
- Exiting from standby mode.

These options allow the user to change values without affecting the immediate image processing, avoiding image corruption.

Examples of Switching from One Context to Another

[Go to Context A]

VAR8 = 1, 0x00, 0x01 // SEQ_CMD

Poll VAR8 = 1, 0x01, 0x03 // SEQ_STATE

[Go to Context B]

VAR8 = 1, 0x00, 0x02 // SEQ_CMD

Poll VAR8 = 1, 0x01, 0x07 // SEQ_STATE

[Go to Capture]

VAR8 = 27, 0x0005, 0x00 // PRI_B_NUM_OF_FRAMES_RUN
VAR8 = 1, 0x0000, 0x02 // SEQ_CMD

POLL VAR8 = 1, 0x01, 0x07 // Poll for SEQ_STATE

[Go to Preview]
VAR8 = 27, 0x0005, 0x05 // PRI_B_NUM_OF_FRAMES_RUN
VAR8 = 1, 0x0000, 0x01 // SEQ_CMD

POLL VAR8 = 1, 0x01, 0x03 // Poll for SEQ_STATE

Scaling

Registers linked to scaling can be automatically adjusted by setting the related mode variables. In preview mode, these variables need to be adjusted to 1024 x 768 or lower resolution.

Notes:
1. At all times, the output sizes above must not exceed the input size to the scaled (crop sizes).
2. If the modification in scaling occurs for the current (active) context, a refresh command (sequence variable 0x0000 = 0x6) is needed to reflect the new values. Otherwise, a context switch to the targeted context will automatically refresh in these values.

Examples of .ini Settings for Different Output Resolutions

```
[QVGA]
VAR = 26, 0x00, 0x0140   // PRI_A_IMAGE_WIDTH
VAR = 26, 0x02, 0x00F0   // PRI_A_IMAGE_HEIGHT
VAR = 18, 0x2B, 0x0408   // CAM1_CTX_A_OUTPUT_SIZE_WIDTH
VAR = 18, 0x2D, 0x0308   // CAM1_CTX_A_OUTPUT_SIZE_HEIGHT
VAR8 = 1, 0x00, 0x06   // SEQ_CMD
```

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 same variables used for cropping provide all the zoom functions.

[Digital Zoom Context A – off]

VAR = 23, 0x3, 0x20 // ENABLE ZOOM
VAR = 26, 0x00, 0x0140 // PRI_A_IMAGE_WIDTH
VAR = 26, 0x02, 0x00F0 // PRI_A_IMAGE_HEIGHT
VAR = 23, 0x18, 0x0064 // SYS_ZOOM_FACTOR
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Digital Zoom Context A – x1.6]

VAR = 23, 0x3, 0x20 // ENABLE ZOOM
VAR = 26, 0x00, 0x0140 // PRI_A_IMAGE_WIDTH
VAR = 26, 0x02, 0x00F0 // PRI_A_IMAGE_HEIGHT
VAR = 23, 0x18, 0x00A0 // SYS_ZOOM_FACTOR
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Digital Zoom Context A – x2.4]

VAR = 23, 0x3, 0x20 // ENABLE ZOOM
VAR = 26, 0x00, 0x0140 // PRI_A_IMAGE_WIDTH
VAR = 26, 0x02, 0x00F0 // PRI_A_IMAGE_HEIGHT
VAR = 23, 0x18, 0x00F0 // SYS_ZOOM_FACTOR
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Digital Zoom Context A – x3.2]

VAR = 23, 0x3, 0x20 // ENABLE ZOOM
VAR = 26, 0x00, 0x0140 // PRI_A_IMAGE_WIDTH
VAR = 26, 0x02, 0x00F0 // PRI_A_IMAGE_HEIGHT
VAR = 23, 0x18, 0x0140 // SYS_ZOOM_FACTOR
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Digital Zoom Context B (output size = VGA) x1.6]

VAR = 23, 0x3, 0x20 // ENABLE ZOOM
VAR = 27, 0x00, 0x0280 // PRI_B_IMAGE_WIDTH
VAR = 27, 0x02, 0x01E0 // PRI_B_IMAGE_HEIGHT
VAR = 23, 0x18, 0x00A0 // SYS_ZOOM_FACTOR
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

Enabling Special Effects

Special effects (monochrome, sepia, negative, solarization) formats can be selected by programming the following variables:

- Context A: ID 26, offset 0x83
- Context B: ID 27, offset 0x83

Note: A refresh command is necessary to enable the new settings for special effects.

Examples of Programming for Special Effects

The following settings show how to program the MT9T111 for special effects:

[Special Effect – Black/White]

VAR8 = 26, 0x83, 0x01 // PRI_A_CONFIG_SYSCTRL_SELECT_FX
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Special Effect – Negative]

VAR8 = 26, 0x83, 0x03 // PRI_A_CONFIG_SYSCTRL_SELECT_FX
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Special Effect – Solarize w/ Strength Control]

VAR8 = 26, 0x83, 0x04 // PRI_A_CONFIG_SYSCTRL_SELECT_FX
VAR8 = 26, 0x84, 0x08 // PRI_A_CONFIG_SYSCTRL_SOLARIZATION_TH
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Special Effect – Sepia w/ Tint Control]

VAR8 = 26, 0x83, 0x02 // PRI_A_CONFIG_SYSCTRL_SELECT_FX
VAR8 = 26, 0x85, 0x30 // PRI_A_CONFIG_SYSCTRL_SEPIA_CR
VAR8 = 26, 0x86, 0xA0 // PRI_A_CONFIG_SYSCTRL_SEPIA_CB
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

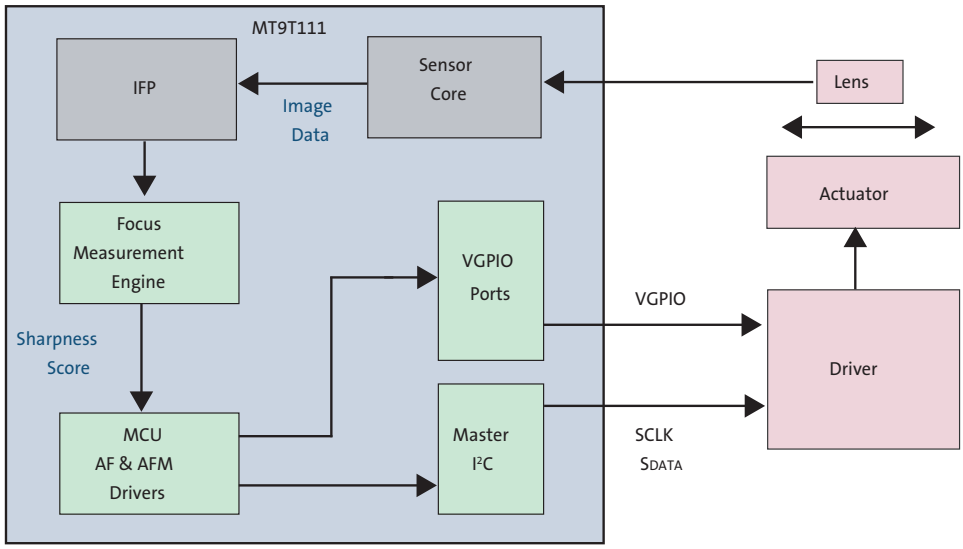
[Special Effect – Off]

VAR8 = 26, 0x83, 0x00 // PRI_A_CONFIG_SYSCTRL_SELECT_FX
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

Auto Focus

The AF algorithm implemented in the MT9T111 seeks to maximize the sharpness of vertical lines in the sensor's image output by guiding an external lens actuator to the position of the best lens focus. The algorithm's implementation has a hardware component called the focus measurement engine (FME) and a firmware component called AF driver. The algorithm is lens-actuator-independent: it provides guidance by means of an abstract 8-bit variable called the logical_lens_position, leaving the translation of its changes into physical lens movements to a separate auto focus mechanics (AFM) driver, as shown in Figure 51.

Figure 51: Auto Focus Functional Block Diagram



AF Algorithm

The AF algorithm relies on the auto focus mechanics (AFM) driver and the master two-wire serial interface (I^2C-type driver) or VGPIO driver to generate digital output signals needed to move different lens actuators. The MT9T111 supports various types of actuators including both micro-electro mechanical systems (MEMS) and voice-coil motor (VCM) types of actuators. The MT9T111 does not support stepping motor type actuators. The AFM driver must correctly indicate at all times if the lens it controls is stationary or moving. This is required to prevent the AF driver from using line sharpness measurements distorted by concurrent lens motion, and from issuing new commands to move the lens while the previous one is still being executed.

Table 34 on page 92 shows the driver ICs supported by the MT9T111. Users should program the AFM variable offset 0x0003[2:0] to select the driver IC that is connected to MT9T111. AFM variable offset 0x0021 stores the slave address for I^2C type of driver IC.

In addition to those drives indicated in Table 34 on page 92, the MT9T111 supports other types of driver interfaces through the use of customized code.

Table 34: Auto Focus ICs Supported by the MT9T111

Driver IC	Interface	Sensor Rev2	Sensor Rev3	0x0003[2:0]
AD5398	I^2C	Yes	Yes	1
DW9710	I^2C	Yes	Yes	1
MD118B	I^2C	Yes	Yes	1
MIC2290	PWM	No	Yes	4
LT3469	PWM	No	Yes	4
D88	PWM	No	Yes	4

AF Mode

The MT9T111 supports five different modes for AF operation. Figure 52 and Figure 53 on page 93 describe the full scan mode and hill climbing mode. Continuous mode uses the motion detection block for triggering auto focus and uses the hill climbing mode. The user can monitor the best position register located in the AF variable offset 0x0024 to monitor the lens position movement.

- Manual mode: (AF offset 0x03[0])
- Full scan mode: (AF offset 0x03[1])
- Creep compensation mode: (AF offset 0x03[2])
- Hill climbing mode: (AF offset 0x03[3])
- Continuous AF mode: (AF offset 0x03[4])

Figure 52: Full-Scan Mode AF

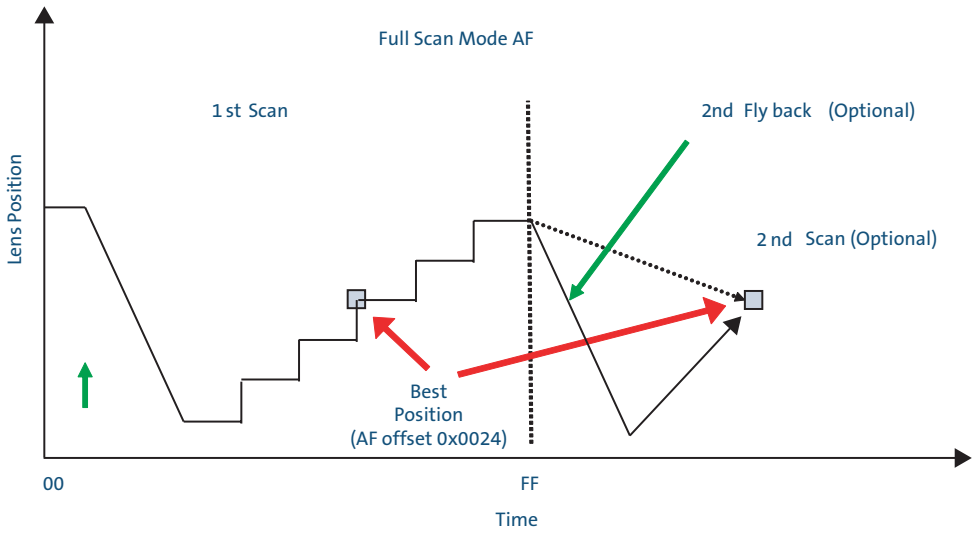
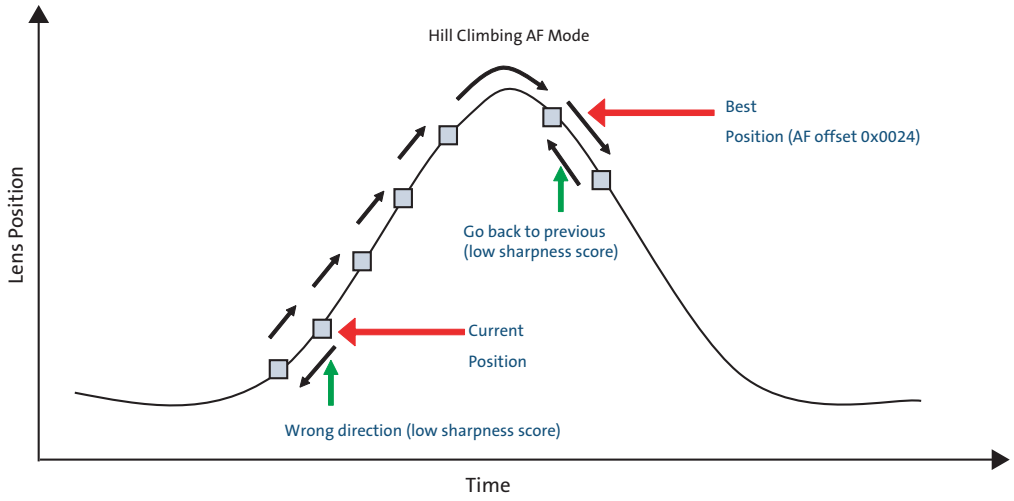


Figure 53: Hill Climbing AF Mode



Example of Programming Simple Full-Scan Triggering Operation

The following programming shows the simple full scan triggering operation for the MT9T111.

```
REG = 0x0614, 0x0000  //  Needed for Rev1 and Rev2 to activate SDA pad
REG = 0x98E, 0x4403  //  AFM Variable offset 0x0003
REG = 0x990, 0x8001  //  Set [15] = 1 and [0] = 1
```

Note: [15] = “0” indicates loading is completed.

```
REG = 0x98E, 0xC421  //  AFM variable offset 0x0021
REG = 0x990, 0x18  //  Driver IC I2C address = 0x18
REG = 0x98E, 0x3003  //  AF Variable offset 0x0003
REG = 0x990, 0x0002  //  set [1] = 1 to enter full scan operation
REG = 0x98E, 0xB019  //  AF Variable offset 0x0019
REG = 0x990, 0x01  //  set [0] = 1 to trigger
REG = 0x98E, 0x8400  //  Sequencer Variable offset 0x0000
REG = 0x990, 0x05  //  Refresh command
```

Anti-Shake

Introduction

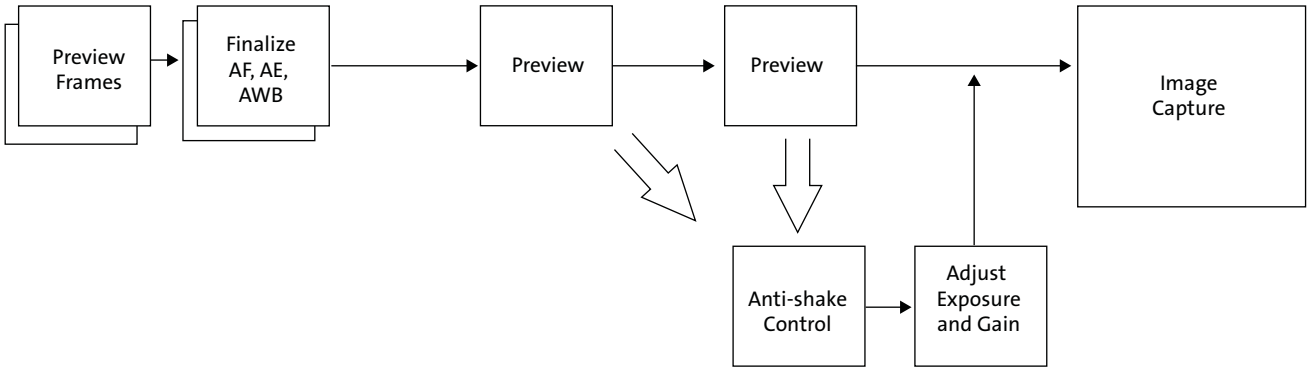
As mobile devices become smaller, unavoidable handshaking or moving subjects make it difficult for a user to hold a slim mobile camera steadily enough to get a flawless image, especially when exposure time increases due to low-light conditions. To reduce motion blur, the MT9T111 includes an anti-shake mode. When motion is detected, the anti-shake controller will increase the sensor gain and reduce the exposure time. The anti-shake control reduces the motion blur caused by a shaking camera.

The anti-shake control runs only when transitioning from a preview context to a capture context, and cannot be used during streaming preview for stabilized viewfinder or video applications.

Algorithm Description

The anti-shake algorithm uses motion detection statistics from the sensor. The anti-shake algorithm (as shown graphically in Figure 54) uses two successive image frames to compute a motion estimate. The two frames are not output from the camera; only the statistics are output. The two frames are exposed at the fastest speed possible, given the current total exposure and camera gain constraints. The image size is constrained to be VGA or greater for this operation.

Figure 54: Anti-Shake Algorithm



The anti-shake algorithm is launched when transitioning from preview to capture. The algorithm assumes control from the sequencer when switching between these modes. Settings for AE, AF, and AWB are finalized during preview.

During the first frame, the algorithm identifies eight regions of interest for which to compute motion. During the second frame, the algorithm computes motion for the identified regions of interest. Based on these motion estimates, the algorithm determines the optimal balance of exposure and gain for the scene.

After determining the combination of gain and exposure, the algorithm returns control to the sequencer, which configures the camera for capture. Following the capture, the camera is returned to the original gain and exposure combination to minimize visual transition effects.

Configuration

The anti-shake algorithm has two major controls: one that controls feature thresholds for determining regions of interest, and another to control the target motion blur in those regions. These controls, along with the integration time scale factor and min integration time for the first AS frame can be set directly by setting pri_a_config_is_mode[3:0].

Alternately, the value can be calculated by the firmware by interpolating between appropriate start and stop values based on a range of brightness metric's start and stop values. This is achieved by setting pri_a_config_is_mode[3] to "1". The user can then use pri_a_config_is_mode[2] to decide if the coarse integration time calculated by the anti-shake function obeys the flicker periods.

The value of the feature threshold variable, pri_a_config_is_feature_threshold, determines the threshold for feature selection. A higher value commands the algorithm to identify sharper features in the image on which to compute motion. This variable ranges from 0x00 to 0xFF, but is usually set to a value between 0x0A and 0x28.

The value of the desired blur parameter, pri_a_config_is_blur_input_parameter, determines the desired blur in the captured image. If the target is greater than the lowest blur achievable for the current motion, this value will be used to determine the blur in the captured image. This value typically ranges between 0x03 for sharp images, and 0x07 for images with less noise.

The value for maximum digital gain, cam1_as_max_digital_gain_allowed, limits the digital gain applied to the image. This value typically ranges from 0x80 for images with less noise to 0xFF for sharp images.

To turn on the anti-shake function:

1. Set driver ID = 26 // cam_pricontext_a .
2. Set the offset 0x56 = 0x15.
3. Set VAR8 = 26, 0x56, 0x15 // pri_a_config_is_mode.

Example of Settings for Anti-Shake

The following example settings show the registers related to AS control in the MT9T111 when pri_a_config_is_mode[3] = 1.

[Anti-Shake - Normal]

```
VAR = 18, 0x010E, 0x00A4 // CAM1_AS_MAX_DIGITAL_GAIN_ALLOWED
VAR8 = 13, 0x003D, 0x31 // AS_START_ASVALUES_0
VAR8 = 13, 0x003E, 0x1B // AS_START_ASVALUES_1
VAR8 = 13, 0x003F, 0x28 // AS_START_ASVALUES_2
VAR8 = 13, 0x0040, 0x03 // AS_START_ASVALUES_3
VAR8= 13, 0x0041, 0xCD // AS_STOP_ASVALUES_0
VAR8 = 13, 0x0042, 0x64 // AS_STOP_ASVALUES_1
VAR8 = 13, 0x0043, 0x0F // AS_STOP_ASVALUES_2
VAR8 = 13, 0x0044, 0x07 // AS_STOP_ASVALUES_3
```

[Anti-Shake - Less Noise]

```
VAR = 18, 0x010E, 0x0080 // CAM1_AS_MAX_DIGITAL_GAIN_ALLOWED
VAR8 = 13, 0x003D, 0x31 // AS_START_ASVALUES_0
VAR8 = 13, 0x003E, 0x1B // AS_START_ASVALUES_1
VAR8 = 13, 0x003F, 0x28 // AS_START_ASVALUES_2
VAR8 = 13, 0x0040, 0x04 // AS_START_ASVALUES_3
```

VAR8 = 13, 0x0041, 0xCD // AS_STOP_ASVALUES_0
VAR8 = 13, 0x0042, 0x64 // AS_STOP_ASVALUES_1
VAR8 = 13, 0x0043, 0x0F // AS_STOP_ASVALUES_2
VAR8 = 13, 0x0044, 0x08 // AS_STOP_ASVALUES_3

[Anti-Shake - Sharper]

VAR = 18, 0x010E, 0x0100 // CAM1_AS_MAX_DIGITAL_GAIN_ALLOWED
VAR8 = 13, 0x003D, 0x31 // AS_START_ASVALUES_0
VAR8 = 13, 0x003E, 0x1B // AS_START_ASVALUES_1
VAR8 = 13, 0x003F, 0x28 // AS_START_ASVALUES_2
VAR8 = 13, 0x0040, 0x02 // AS_START_ASVALUES_3
VAR8 = 13, 0x0041, 0xCD // AS_STOP_ASVALUES_0
VAR8 = 13, 0x0042, 0x64 // AS_STOP_ASVALUES_1
VAR8 = 13, 0x0043, 0x0F // AS_STOP_ASVALUES_2
VAR8 = 13, 0x0044, 0x06 // AS_STOP_ASVALUES_3

Lens Shading Correction (LC)

Lenses and other factors cause fixed pattern signal gradients in images. The cumulative result of these factors is known as image shading. The MT9T111 has an embedded shading correction module that can be programmed to counter the shading effects on each individual R, Gb, Gr, and B color signal. The shading correction can correct for complex shading beyond the radially symmetric shading from lens falloff.

The MT9T111 implements a digital algorithm for shading correction that addresses these complex shading effects. This algorithm is capable of correcting asymmetric shading in addition to the circularly symmetric lens falloff.

The shading correction registers (R0x3640–R0x3684) must be programmed with the center of the image and a correction function. The center of the image is defined as the point on the array aligned with the optical axis of the lens. This center can be automatically determined during calibration.

The correction curve will be computed during calibration. It can be measured by varying criteria, including response flatness across the image, or on specified lines through the center of the image, color separation across the image, or purely visual criteria. Selection of calibration parameters should consider the evaluation criteria. While response flatness across the image may seem a desirable goal, the actual implementation may result in over-correction and enhanced noise in the corners. Reducing the target response in the corners can result in improved color performance and more natural images.

Refer to "Lens Calibration Procedure" on page 122 in the "Development Tool Overview" on page 115 for a more detail description.

Related Registers for the Lens Shading Algorithm

PGA settings are stored in shading correction registers from PGA1 through PGA102 (0x3640 –0x3784).

The lens shading algorithm can be enabled or disabled through the SOC1 register map 0x3210[3]. To calibrate, set the sensor to output Bayer full resolution and disable the MCU.

Example: PGA Values for LC

The following example settings show the registers containing PGA values in the MT9T111.

[Lens Correction]

REG = 0x364A, 0x0310 // P_R_P0Q0
REG = 0x364C, 0x1E6C // P_R_P0Q1
REG = 0x364E, 0x0CF2 // P_R_P0Q2
REG = 0x3650, 0x74ED // P_R_P0Q3
REG = 0x3652, 0x56F1 // P_R_P0Q4
REG = 0x368A, 0x304D // P_R_P1Q0
REG = 0x368C, 0x510C // P_R_P1Q1
REG = 0x368E, 0x1210 // P_R_P1Q2
REG = 0x3690, 0x880F // P_R_P1Q3
REG = 0x3692, 0x9AF1 // P_R_P1Q4
REG = 0x36CA, 0x5B92 // P_R_P2Q0
REG = 0x36CC, 0xDA6F // P_R_P2Q1
REG = 0x36CE, 0x7693 // P_R_P2Q2

REG = 0x36D0, 0xD491 // P_R_P2Q3
REG = 0x36D2, 0x9676 // P_R_P2Q4
REG = 0x370A, 0x4130 // P_R_P3Q0
REG = 0x370C, 0x5770 // P_R_P3Q1
REG = 0x370E, 0x8D14 // P_R_P3Q2
REG = 0x3710, 0x9251 // P_R_P3Q3
REG = 0x3712, 0x32F6 // P_R_P3Q4
REG = 0x374A, 0x4D12 // P_R_P4Q0
REG = 0x374C, 0xEDB0 // P_R_P4Q1
REG = 0x374E, 0x9377 // P_R_P4Q2
REG = 0x3750, 0xE134 // P_R_P4Q3
REG = 0x3752, 0x7619 // P_R_P4Q4
REG = 0x3640, 0x04F0 // P_G1_P0Q0
REG = 0x3642, 0x050A // P_G1_P0Q1
REG = 0x3644, 0x2E91 // P_G1_P0Q2
REG = 0x3646, 0x29AD // P_G1_P0Q3
REG = 0x3648, 0x5531 // P_G1_P0Q4
REG = 0x3680, 0x616C // P_G1_P1Q0
REG = 0x3682, 0x100C // P_G1_P1Q1
REG = 0x3684, 0x06F0 // P_G1_P1Q2
REG = 0x3686, 0xCA0B // P_G1_P1Q3
REG = 0x3688, 0xE491 // P_G1_P1Q4
REG = 0x36C0, 0x19D2 // P_G1_P2Q0
REG = 0x36C2, 0x9630 // P_G1_P2Q1
REG = 0x36C4, 0x6010 // P_G1_P2Q2
REG = 0x36C6, 0x9F0D // P_G1_P2Q3
REG = 0x36C8, 0x1CD1 // P_G1_P2Q4
REG = 0x3700, 0x554F // P_G1_P3Q0
REG = 0x3702, 0x03D0 // P_G1_P3Q1
REG = 0x3704, 0xA4F3 // P_G1_P3Q2
REG = 0x3706, 0x8B91 // P_G1_P3Q3
REG = 0x3708, 0x7035 // P_G1_P3Q4
REG = 0x3740, 0x1C52 // P_G1_P4Q0
REG = 0x3742, 0xD330 // P_G1_P4Q1
REG = 0x3744, 0xC634 // P_G1_P4Q2
REG = 0x3746, 0xA914 // P_G1_P4Q3
REG = 0x3748, 0x51F8 // P_G1_P4Q4
REG = 0x3654, 0x02F0 // P_B_P0Q0
REG = 0x3656, 0x712A // P_B_P0Q1
REG = 0x3658, 0x0651 // P_B_P0Q2
REG = 0x365A, 0x0BAE // P_B_P0Q3
REG = 0x365C, 0x2DB2 // P_B_P0Q4
REG = 0x3694, 0x8D2A // P_B_P1Q0
REG = 0x3696, 0x45ED // P_B_P1Q1
REG = 0x3698, 0x372F // P_B_P1Q2
REG = 0x369A, 0x8E4F // P_B_P1Q3
REG = 0x369C, 0x8F31 // P_B_P1Q4
REG = 0x36D4, 0x7C11 // P_B_P2Q0
REG = 0x36D6, 0x8CD0 // P_B_P2Q1
REG = 0x36D8, 0x2CF3 // P_B_P2Q
REG = 0x36DA, 0x6E0B // P_B_P2Q3
REG = 0x36DC, 0x8075 // P_B_P2Q4
REG = 0x3714, 0x776F // P_B_P3Q0

```
REG = 0x3716, 0x3410  //  P_B_P3Q1
REG = 0x3718, 0x96B3  //  P_B_P3Q2
REG = 0x371A, 0xF0F0  //  P_B_P3Q3
REG = 0x371C, 0x6935  //  P_B_P3Q4
REG = 0x3754, 0x46D3  //  P_B_P4Q0
REG = 0x3756, 0xB611  //  P_B_P4Q1
REG = 0x3758, 0x8596  //  P_B_P4Q2
REG = 0x375A, 0xA654  //  P_B_P4Q3
REG = 0x375C, 0x7E58  //  P_B_P4Q4
REG = 0x365E, 0x0270  //  P_G2_P0Q0
REG = 0x3660, 0x196A  //  P_G2_P0Q1
REG = 0x3662, 0x2D51  //  P_G2_P0Q2
REG = 0x3664, 0x524C  //  P_G2_P0Q3
REG = 0x3666, 0x45F1  //  P_G2_P0Q4
REG = 0x369E, 0x72A8  //  P_G2_P1Q0
REG = 0x36A0, 0x444D  //  P_G2_P1Q1
REG = 0x36A2, 0x342F  //  P_G2_P1Q2
REG = 0x36A4, 0xF54E  //  P_G2_P1Q3
REG = 0x36A6, 0xC7F1  //  P_G2_P1Q4
REG = 0x36DE, 0x2152  //  P_G2_P2Q0
REG = 0x36E0, 0xBD10  //  P_G2_P2Q1
REG = 0x36E2, 0x1D91  //  P_G2_P2Q2
REG = 0x36E4, 0x1350  //  P_G2_P2Q3
REG = 0x36E6, 0x88CF  //  P_G2_P2Q4
REG = 0x371E, 0x42EF  //  P_G2_P3Q0
REG = 0x3720, 0x46EF  //  P_G2_P3Q1
REG = 0x3722, 0xEFD2  //  P_G2_P3Q2
REG = 0x3724, 0x3A0E  //  P_G2_P3Q3
REG = 0x3726, 0x5C55  //  P_G2_P3Q4
REG = 0x375E, 0x43B2  //  P_G2_P4Q0
REG = 0x3760, 0xFDED  //  P_G2_P4Q1
REG = 0x3762, 0x9DB5  //  P_G2_P4Q2
REG = 0x3764, 0x8095  //  P_G2_P4Q3
REG = 0x3766, 0x6318  //  P_G2_P4Q4
REG = 0x3784, 0x045C  //  CENTER_COLUMN
REG = 0x3782, 0x0310  // CENTER_ROW
STATE = Lens Correction Falloff, 95
STATE = Lens Correction Center X, 1116
STATE = Lens Correction Center Y, 776
BITFIELD = 0x3210, 0x0008, 1  // PGA_ENABLE
```

Auto White Balance (AWB)

This section discusses the procedures to use when programming the color correction matrix elements and manual 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 AWB algorithm.

Color Correction Procedure

Color correction is done after the second black level block, and before the aperture correction block. The interpolated RGB values are transformed by the color correction matrix into color-corrected R', G', and B' values. The color correction matrix is uploaded from the AWB driver into the corresponding registers in the color pipeline when AWB has settled.

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.” All the color correction matrix values are stored in the AWB variable map.

Table 35: Color Correction Matrix Structure

	Gain R	Gain G	Gain B
Saturation red	ccmL[0]	ccmL[1]	ccmL[2]
Saturation green	ccmL[3]	ccmL[4]	ccmL[5]
Saturation blue	ccmL[6]	ccmL[7]	ccmL[8]

AWB Procedure

The AWB algorithm is designed to compensate for the effects of changing spectra of the scene illumination on the quality of the color rendition.

The algorithm consists of two major parts:

- A measurement engine performing statistical analysis of the image.
- The AWB driver performing the selection of the optimal color correction matrix, digital, and sensor core analog gains.

The driver keeps analog gain ratio of left and right matrices corresponding to two opposite illuminations: red-rich (incandescent) and blue-rich (daylight), respectively. The AWB driver analyzes measurement engine data and sets appropriate digital AWB gains and current matrix position (0x0053). The color correction matrix position defines the current matrix coefficients and analog gain ratio.

Example: CCM Values for AWB

The following example settings show the registers related to AWB control in the MT9T111.

[Enable AWB in Context A]

```
VAR = 26, 0x3D, 0x0000  // PRI_A_CONFIG_AWB_ALGO_ENTER
VAR = 26, 0x3F, 0x00FF  // PRI_A_CONFIG_AWB_ALGO_RUN
VAR = 26, 0x41, 0x0000  // PRI_A_CONFIG_AWB_ALGO_LEAVE
VAR8 = 1, 0x00, 0x06  // SEQ_CMD
```

[Enable Manual WB in Context A]

```
VAR = 26, 0x3D, 0x0000  // PRI_A_CONFIG_AWB_ALGO_ENTER
VAR = 26, 0x3F, 0x00F7  // PRI_A_CONFIG_AWB_ALGO_RUN
VAR = 26, 0x41, 0x0000  // PRI_A_CONFIG_AWB_ALGO_LEAVE
VAR8 = 11, 0x35, 0x75  // AWB_CCMPOSITION -->Manual WB control

// Color fine tuning

VAR = 26, 0x43, 0x0003  // PRI_A_CONFIG_AWB_AWB_XSHIFT
VAR = 26, 0x45, 0x0010  // PRI_A_CONFIG_AWB_AWB_YSHIFT
VAR8 = 26, 0x49, 0x40  // PRI_A_CONFIG_AWB_X0
VAR8 = 26, 0x4A, 0x80  // PRI_A_CONFIG_AWB_K_R_L
VAR8 = 26, 0x4B, 0x80  // PRI_A_CONFIG_AWB_K_G_L
VAR8 = 26, 0x4C, 0x80  // PRI_A_CONFIG_AWB_K_B_L
VAR8 = 26, 0x4D, 0x80  // PRI_A_CONFIG_AWB_K_R_R
VAR8 = 26, 0x4E, 0x80  // PRI_A_CONFIG_AWB_K_G_R
VAR8 = 26, 0x4F, 0x80  // PRI_A_CONFIG_AWB_K_B_R

VAR8 = 1, 0x00, 0x06  // SEQ_CMD
```

[Register A light CCM]

```
REG  =  0x32C0, 0x3923   //  (1) COLOR_CORR_MATRIX_SCALE_14
REG  =  0x32C2, 0x0724   //  (1) COLOR_CORR_MATRIX_SCALE_11
REG  =  0x32C4, 0x86C0   //  (2) COLOR_CORR_MATRIX_1_2
REG  =  0x32C6, 0x3618   //  (2) COLOR_CORR_MATRIX_3_4
REG  =  0x32C8, 0x34BE   //  (2) COLOR_CORR_MATRIX_5_6
REG  =  0x32CA, 0xE10C   //  (2) COLOR_CORR_MATRIX_7_8
REG  =  0x32CC, 0x2DF4   //  (2) COLOR_CORR_MATRIX_9
```

[Register D65 light CCM]

```
REG = 0x3012, 0x0380   // (8) COARSE_INTEGRATION_TIME
REG = 0x32C0, 0x3923   // (1) COLOR_CORR_MATRIX_SCALE_14
REG = 0x32C2, 0x0724   // (1) COLOR_CORR_MATRIX_SCALE_11
REG = 0x32C4, 0xD9FC   // (2) COLOR_CORR_MATRIX_1_2
REG = 0x32C6, 0x0306   // (2) COLOR_CORR_MATRIX_3_4
REG = 0x32C8, 0x89D3   // (2) COLOR_CORR_MATRIX_5_6
REG = 0x32CA, 0x9622   // (2) COLOR_CORR_MATRIX_7_8
REG = 0x32CC, 0x2FC7   // (2) COLOR_CORR_MATRIX_9
```

Auto Exposure (AE)

Introduction

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

AE is implemented by a firmware driver that analyzes image statistics collected by the exposure measurement engine. It makes a decision and programs the sensor core and color pipeline to achieve the desired exposure. The measurement engine subdivides the image into 16 windows organized as a 4 x 4 grid.

Two AE algorithm modes are available:

- Average brightness tracking (ABT)
- Dynamic range tracking (DRT)

The average brightness tracking AE uses a constant average tracking algorithm where a target brightness value is compared to a current brightness value, and the gain and integration time are adjusted accordingly to meet the target requirement.

The dynamic range tracking AE examines the data from the statistics engine and produces a corrected brightness target so that overexposed or underexposed scenes can be avoided. This also allows a manual control of the image brightness. Both algorithms are used in both preview and capture modes.

AE Driver

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

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 speed of response, and control of the sensitivity to the small changes. While the default settings are adequate in most situations, the user can program target brightness, measurement window, and other parameters described above.

The driver calculates image brightness based on average luma values received from 16 programmable, equal-size rectangular windows forming a 4 x 4 grid. In preview mode, sixteen windows are combined in two segments: central and peripheral. The central segment includes 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 driver changes AE parameters (integration time, gains, and so on) to drive brightness to the programmable target. The value of the single step approach to the target value can be controlled.

To avoid unwanted reaction of AE on small fluctuations of scene brightness or momentary scene changes, the AE driver uses a temporal filter for luma and a threshold around the AE luma target. The driver changes AE parameters only if the buffered luma is larger than the AE target step and pushes the luma beyond the threshold.

Evaluation Algorithm

A scene-evaluation 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 the original exposure target. It is most useful for backlight and bright outdoor conditions.

Accelerated Settling During Overexposure

The AE speed is direction-dependent. Transitioning from over saturation to target can take more time than transitioning from under saturation. The AE driver has a mode that speeds up AE for overexposed scenes.The AE driver counts the number of AE windows whose average brightness is equal to or greater than some value, 250 by default. For a scene having saturated regions, the average luma is underestimated due to signal clipping. The driver compensates underestimation by a factor that can be user-defined.

Exposure Control

To achieve the required amount of exposure, the AE driver adjusts the sensor integration time, gains, ADC reference, and IFP digital gains. In addition, a variable is available for the user to adjust the overall brightness of the scene. To reject flicker, integration time is typically adjusted in increments of steps. The incremental step specifies the duration in row times equal to one flicker period. Thus, flicker is rejected if integration time is kept a natural factor of the flicker period.

Configuration

The AE algorithm consists of two parts: a set of rules that decides what average brightness target would be appropriate for the current scene based on statistics, and a tracking algorithm that adjusts the exposure time and gain to the achieve the target brightness.

The tracking driver uses the luminance target computed by the rule driver. This target can be read at ae_track_target. The host can adjust the setting of ae_track_gate to avoid unwanted reaction to small fluctuations of scene brightness. The ae_track_gate specifies a gate around the target luminance. If the current scene's brightness is within this gate, then the AE algorithm will not attempt to adjust the scene's brightness. The variable ae_track_current_y indicates the current brightness in the scene. This can be compared to ae_track_target and ae_track_gate to determine if the scene brightness has converged to the target. The ae_tracking_dampening variable dampens the rate at which the scene brightness changes by averaging the current scene's brightness with the dampened average of previous frames.

The rules driver is controlled by the variable ae_rule_algo. This variable contains individual flags that control which rule sets are applied to the current exposure. The flag ae_rule_exec_detect_backlight executes a method that detects backlight. The flag ae_rule_exec_rule_avgy bypasses the rules, and should not be set if ae_rule_exec_rule_drt is set.

The user should set these through the PRI_A and PRI_B pages, not in the driver. They will be deleted during context switches if set only in the driver (this is true of all driver algorithm settings).

The flag ae_rule_exec_rule_drt of variable ae_rule_algo enables the rule checking. These rules will raise or lower the brightness target depending on the shape of the histogram, and adjust the brightness target if the driver detects backlight.

Aside from configuring the rules and tracking, basic AE variables can be configured by context variables. The variable pri_a_config_ae_rule_base_target sets the base target brightness for a scene. The rules driver will modify this base target. The variables pri_a_config_ae_track_ae_min_virt_dgain and pri_a_config_ae_track_ae_max_virt_dgain limit the digital gains applied by the tracking driver. The variables pri_a_config_ae_track_ae_min_virt_again and pri_a_config_ae_track_ae_max_virt_again limit the analog gains applied by the tracking driver.

Related Registers

- ae_track_target_fdzone (0x002D of pri_a and pri_b)
  The AE algorithm tries to keep the integration time below the programmed value in 0x002D by giving priority to increasing gain first
- ae_track_target_again (0x002F of pri_a and pri_b)
  The AE algorithm tries to keep the gain below the programmed value in 0x002F by giving priority to increasing integration time first
- ae_track_max_fdzone_50Hz (0x0015 of pri_a and pri_b)
- ae_track_max_fdzone_60Hz (0x0017 of pri_a and pri_b)
- Maximum fd zone for integration time
  Specifies lowest frame rate (fps is different between 50Hz and 60Hz flicker setting)

Example: AE Control

The following example settings show the registers related to AE control in the MT9T111.

[Enable AE in context B]

```
REG = 0x98E, 0x6C1B   //  MCU_ADDRESS
REG = 0x990, 0x0008   //  PRI_B_CONFIG_AE_RULE_ALGO_RUN
REG = 0x98E, 0x6C27   //  MCU_ADDRESS
REG = 0x990, 0x0007   //  PRI_B_CONFIG_AE_TRACK_ALGO_RUN
REG = 0x98E, 0x6C5E   //  MCU_ADDRESS
REG = 0x990, 0x801F   //  PRI_B_CONFIG_STAT_ALGO_ENTER
REG = 0x98E, 0x6C60   //  MCU_ADDRESS
REG = 0x990, 0x12E0   //  PRI_B_CONFIG_STAT_ALGO_RUN
REG = 0x98E, 0x8400   //  MCU_ADDRESS
REG = 0x990, 0x06   //  SEQ_CMD
```

[Disable AE in context B]

```
REG = 0x98E, 0x6C1B   //  MCU_ADDRESS
REG = 0x990, 0x0000   //  PRI_B_CONFIG_AE_RULE_ALGO_RUN
REG = 0x98E, 0x6C27   //  MCU_ADDRESS
REG = 0x990, 0x0000   //  PRI_B_CONFIG_AE_TRACK_ALGO_RUN
REG = 0x98E, 0x6C5E   //  MCU_ADDRESS
REG = 0x990, 0x0000   //  PRI_B_CONFIG_STAT_ALGO_ENTER
REG = 0x98E, 0x6C60   //  MCU_ADDRESS
REG = 0x990, 0x0000   //  PRI_B_CONFIG_STAT_ALGO_RUN
REG = 0x98E, 0x8400   //  MCU_ADDRESS
REG = 0x990, 0x06   //  SEQ_CMD
```

[Brightness_Metric]

//Brightness Metric

VAR8 = 18, 0x0113, 0x0A // CAM1_STAT_BRIGHTNESS_METRIC_PREDIVIDER
VAR = 26, 0x006B, 0x0046 // PRI_A_CONFIG_LL_START_BRIGHTNESS
VAR = 26, 0x006D, 0x012C // PRI_A_CONFIG_LL_STOP_BRIGHTNESS
VAR = 27, 0x006B, 0x0046 // PRI_B_CONFIG_LL_START_BRIGHTNESS
VAR = 27, 0x006D, 0x012C // PRI_B_CONFIG_LL_STOP_BRIGHTNESS
VAR = 13, 0x0039, 0x0046 // AS_ASSTART_BRIGHTNESS
VAR = 13, 0x003B, 0x012C / / AS_ASSTOP_BRIGHTNESS
VAR = 18, 0x0126, 0x0001 // CAM1_LL_START_GAMMA_BM
VAR = 18, 0x0128, 0x0002 // CAM1_LL_MID_GAMMA_BM
VAR = 18, 0x012A, 0x012C // CAM1_LL_STOP_GAMMA_BM
VAR = 19, 0x0126, 0x0001 // CAM2_LL_START_GAMMA_BM
VAR = 19, 0x0128, 0x0002 // CAM2_LL_MID_GAMMA_BM
VAR = 19, 0x012A, 0x012C // CAM2_LL_STOP_GAMMA_BM

[Enable Aperture Correction in Context A]

REG = 0x98E, 0x6867 // MCU_ADDRESS
REG = 0x990, 0x007D // PRI_A_CONFIG_LL_ALGO_RUN
REG = 0x98E, 0x8400 // MCU_ADDRESS
REG = 0x990, 0x06 // SEQ_CMD

[Disable Aperture Correction in Context A]

REG = 0x98E, 0x6867 // MCU_ADDRESS
REG = 0x990, 0x006D // PRI_A_CONFIG_LL_ALGO_RUN
REG = 0x98E, 0x8400 // MCU_ADDRESS
REG = 0x990, 0x06 // SEQ_CMD

DELAY = 50

REG = 0x3210, 0x01A8 // COLOR_PIPELINE_CONTROL

[Enable Aperture Correction in Context B]

REG = 0x98E, 0x6C67 // MCU_ADDRESS
REG = 0x990, 0x0010 // PRI_B_CONFIG_LL_ALGO_RUN
REG = 0x3210, 0x01B8 // COLOR_PIPELINE_CONTROL

[Disable Aperture Correction in Context B]

REG = 0x98E, 0x6C67 // MCU_ADDRESS
REG = 0x990, 0x0000 // PRI_B_CONFIG_LL_ALGO_RUN
REG = 0x98E, 0x8400 // MCU_ADDRESS
REG = 0x990, 0x06 // SEQ_CMD

DELAY = 200

REG = 0x3210, 0x01A8 // COLOR_PIPELINE_CONTROL

Flicker Avoidance

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 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, 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, AE must limit the integration time to an integer multiple of the light's flicker period.

By default, the MT9T111 does all of this automatically, ensuring that all exposure times avoid any noticeable light flicker in the scene. The MT9T111 AE algorithm is always setting exposure times to be integer multipliers of either 100Hz or 120Hz. The flicker detection microcontroller driver monitors the incoming frames to detect whether the scene's lighting has changed to the other of the two light sources.

How to Use the Flicker Detection Driver

The intelligence and adjustability of the flicker detection algorithm resides in the microcontroller code. The flicker detection driver's actions may be controlled by the sequencer driver. For each frame, the flicker driver can be instructed to detect the current flicker period and set the current flicker period. The flicker driver can also override the selected value. These operations are controlled by fd.fd_algo (VAR(0x08, 0x03)), by setting the fd_exec_detectperiod and fd_exec_setperiod flags.

If neither flag is set in fd.fd_algo, the flicker detection is disabled and the value stored in the fd.fd_fdperiod_select is not copied to the configuration variable that controls exposure intervals.

If only the fd_exec_setperiod flag is set in fd.fd_algo, then the flicker value currently stored in fd.fd_fdperiod_select is applied to the AE. This mode disables the automatic analysis of the light source, but allows the user to alter the flicker detection driver variable fd.fd_period to select the light source manually. The value stored in the fd.fd_fdperiod_select is copied to the configuration variable that controls exposure intervals.

If both fd_exec_setperiod and fd_exec_detectperiod flags are set in fd.fd_algo, then the driver 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 configuration variable that controls exposure intervals.

The fd.fd_stat_min and fd.fd_stat_max help control the sensitivity of the algorithm to flickering light in a scene. If the algorithm detects fd.fd_stat_min instances of flickering in fd.fd_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.

The detection intervals for the flicker detection driver will be automatically configured for sensors supported by ROM.

How to Fine-Tune the Anti-Flicker Driver Settings

1. Use Register Wizard to generate the basic anti-flicker settings for the specific timing conditions, including the following:

```
VAR8 = 18, 0x44, 0x5C // CAM1_CTX_A_FDPERIOD_50HZ
VAR8 = 18, 0x45, 0x4C // CAM1_CTX_A_FDPERIOD_60HZ
VAR8 = 18, 0xA5, 0x0C // CAM1_FD_SEARCH_F1_50
VAR8 = 18, 0xA6, 0x0E // CAM1_FD_SEARCH_F2_50
VAR8 = 18, 0xA7, 0x0F // CAM1_FD_SEARCH_F1_60
VAR8 = 18, 0xA8, 0x11 // CAM1_FD_SEARCH_F2_60
```

2. Fine-tune the search window for 50Hz.
   If the auto detection function has a high failure rate, decrease the search_f1_50 value by minus 1, and increase by 1 the search_f2_50 value, to increase the search range. If +/– 1 is not good enough, increase the range accordingly.
3. Fine-tune the search window for 60Hz in the same way as for 50Hz.
4. Increase the sensitivity of the flicker detection.
   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. 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. For most applications, the following steps should produce enough sensitivity. This value can be tuned to suit specific applications.
   VAR8 = 8, 0x09, 0x02 // FD_STAT_MIN
   VAR8 = 8, 0x0A, 0x05 // FD_STAT_MAX
5. Increase the signal strength sensitivity of the flicker detection.
   fd_min_amplitude specifies the signal threshold below which signals are ignored. The following variable setting should be suitable for most cases. This value can be tuned to suit specific applications.
   VAR8 = 8, 0x0C, 0x01 // FD_MIN_AMPLITUDE
6. Call REFRESH or REFRESH_Mode commands to make the above changes effective.

How to Verify the Anti-Flicker Driver Settings

1. Verify the settings for 50Hz manual mode:
   1a. Connect the light source with 50Hz power supply.
   1b. Load the following to enable 50Hz manual mode:
   VAR = 26, 0x11, 0x0002 // PRI_A_CONFIG_FD_ALGO_RUN
   VAR8 = 8, 0x05, 0x01 // FD_FDPERIOD_SELECT
   VAR8 = 1, 0x00, 0x05 // SEQ_CMD
   1c. Check the output image to see if flicker disappears.
   1d. Monitor ae_track_fdperiod (VAR = 10, 0x14) to see if it is equal to cam1_ctx_a_fdperiod_50hz (VAR8 = 18, 0x44)
2. Verify the settings for 60Hz manual mode:
   2a. Connect the light source with 60Hz power supply.
   2b. Load the following to enable 60Hz manual mode:
   VAR = 26, 0x11, 0x0002 // PRI_A_CONFIG_FD_ALGO_RUN
   VAR8 = 8, 0x05, 0x00 // FD_FDPERIOD_SELECT
   VAR8 = 1, 0x00, 0x05 // SEQ_CMD
   2c. Check the output image to see if flicker disappears.
   2d. Monitor ae_track_fdperiod (VAR = 10, 0x14) to see if it is equal to cam1_ctx_a_fdperiod_60hz (VAR8 = 18, 0x45)
3. Verify the setting for auto mode:
   3a. Connect the light source with 60Hz power supply.
   3b. Load the following to enable auto mode:
   VAR = 26, 0x11, 0x0003 // PRI_A_CONFIG_FD_ALGO_RUN
   VAR8 = 1, 0x00, 0x05 // SEQ_CMD
   3c. Check the output image to see if flicker disappears.
   3d. Monitor ae_track_fdperiod (VAR = 10, 0x14) to see if it is equal to cam1_ctx_a_fdperiod_60hz (VAR8 = 18, 0x45)
   3e. Switch the power supply to 50Hz.
   3f. Check the output image to see if flicker disappears.
   3g. Monitor ae_track_fdperiod (VAR = 10, 0x14) to see if it is equal to cam1_ctx_a_fdperiod_50hz (VAR8 = 18, 0x44)

[Frame Rate Control]

VAR = 26, 0x3B, 0x000F
// PRI_A_CONFIG_AE_TRACK_AE_MIN_FDZONE -->Max Frame Rate

VAR = 26, 0x2D, 0x000F
// PRI_A_CONFIG_AE_TRACK_TARGET_FDZONE --> Frame Rate Control Threshold

VAR = 26, 0x15, 0x000E
// PRI_A_CONFIG_FD_MAX_FDZONE_50HZ -->Min Frame Rate for 50Hz Flicker

VAR = 26, 0x17, 0x000F
// PRI_A_CONFIG_FD_MAX_FDZONE_60HZ-->Min Frame Rate for 60Hz Flicker

VAR8 = 1, 0x00, 0x06 // SEQ_CMD

Gamma

The MT9T111 includes a block for gamma correction that can adjust its shape based on brightness to enhance the performance under certain lighting conditions.

Gamma correction takes place in the RGB color space after color correction. The pixel data is mapped to a gamma curve programmed by the user or a curve computed based on the brightness level. The gamma curve is approximated by connecting 18 lines segments. The ends of the line segments are called knee points and are at X coordinates of 0, 64, 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2304, 2560, 2816, 3072, 3328, 3584, 3840, and 4095.

Three gamma correction curves may be uploaded: one corresponding to a brighter lighting condition that will enhance contrast VAR(0x0F, 0x0020–0x0032), another corresponding to a normal lighting condition VAR(0x0F, 0x0033–0x0045), and one corresponding to a darker lighting condition that will suppress noise VAR(0x0F, 0x0046–0x0058). From these three curves, the MT9T111 will compute a gamma curve that is appropriate for the current brightness level. At power-up the MT9T111 loads these three tables with default values.

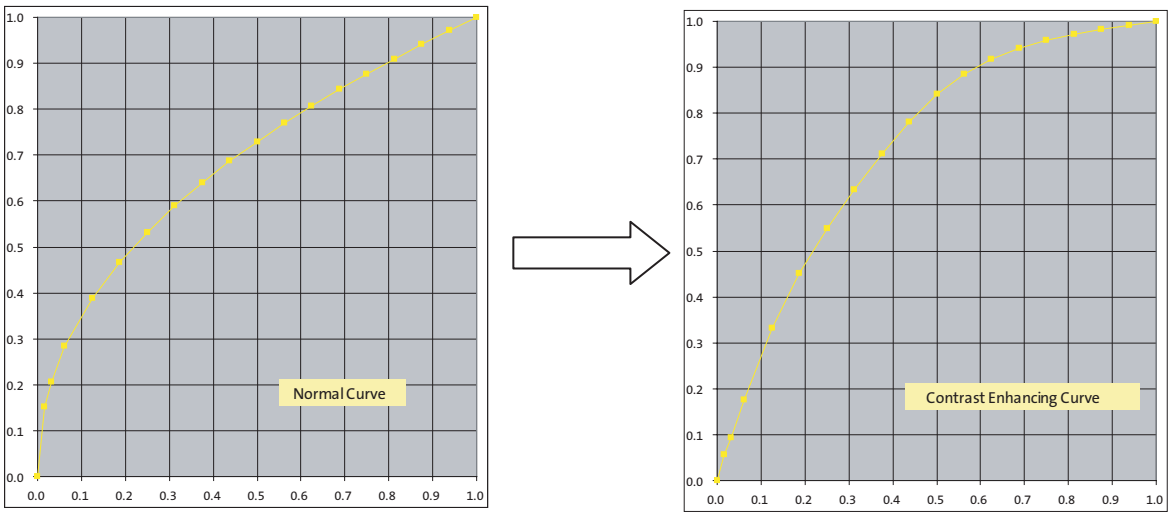
Programming the gamma selection variable VAR(0x0F, 0x0008) to a value of 0x00, automatically calculates a gamma curve from the programmed three curves. Values of 0x01–0x03 select from one of the three programmed curves.

The automatic gamma correction curve is calculated by linearly interpolating each of the 18 knee points in the curve. The linear interpolation is based on the current scene brightness, and three programmed brightness levels corresponding to the three gamma correction curves: VAR(0x12, 0x0169–0x016D). These brightness levels are independent for each sensor, and can be separately set on page 0x13 for the secondary sensor. The computed curve is a linear interpolation of the 18 knee points depending on the current scene brightness.

Bright Scenes

In bright scenes, the gamma function is composed with a contrast enhancement function to obtain the curve on the right of Figure 55. The applied curve will smoothly transition between knee points.

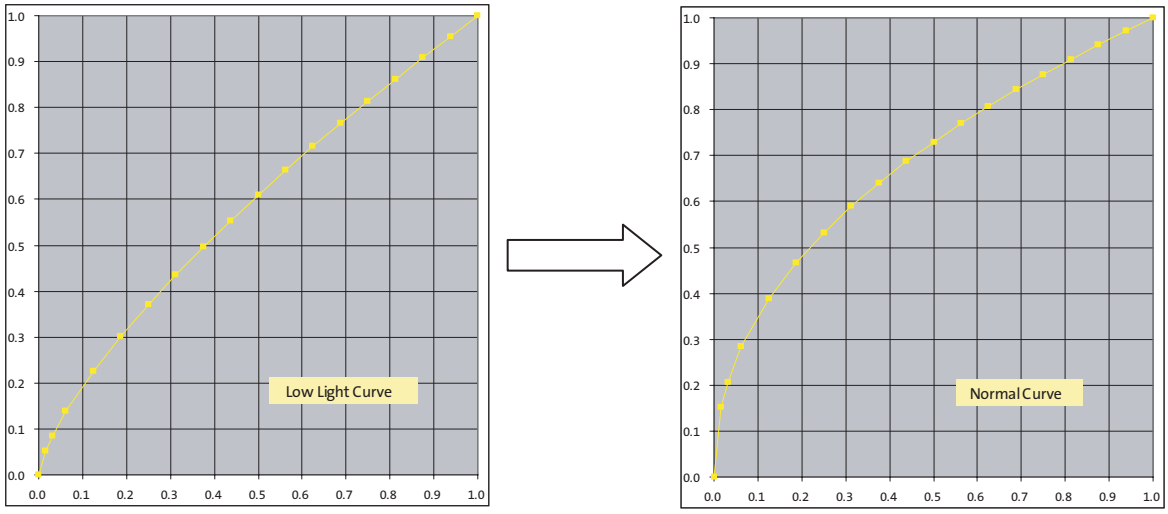
Figure 55: Gamma Correction in Bright Scenes



Dark Scenes

In dark scenes, the gamma function is composed with a noise reduction function to obtain the curve on the right. The applied curve will smoothly transition between knee points, as shown in Figure 56.

Figure 56: Gamma Correction in Dark Scenes



Gamma/Contrast Manual Control

To select manual gamma/contrast curve control the following steps are necessary:

1. Enable manual control of the gamma curve selection by setting driver ID 15, offset 0x09 to 4:
   1a. VAR8 = 15, 0x09, 0x04 // (1) LL_GAMMA_SELECT
2. Select the desired gamma/contrast curve by changing the driver ID 15, offset 0x0A variable to the desired position based on start_gamma_bm, mid_gamma_bm or stop_gamma_bm

Gamma/Contrast Automatic Control

To enable automatic selection, the following steps are necessary:

1. Enable manual control of the gamma curve selection by setting driver ID 15, offset 0x09 (0x08 in rev2) to 0:
   1a. VAR8 = 15, 0x09, 0x00 // (1) LL_GAMMA_SELECT

The gamma curve can be disabled (gamma of 1) by setting 0x3210[7] = 0.

Example: Gamma Control

The following example settings show the registers related to gamma control in the MT9T111.

[Contrast Control]

VAR8 = 15, 0x08, 0x01

// LL_GAMMA_SELECT --> select contrast gamma, change to 0 for auto control

VAR8 = 15, 0x0B, 0x00 // LL_GAMMA_CONTRAST_CURVE_0
VAR8 = 15, 0x0C, 0x0B // LL_GAMMA_CONTRAST_CURVE_1
VAR8 = 15, 0x0D, 0x1F // LL_GAMMA_CONTRAST_CURVE_2
VAR8 = 15, 0x0E, 0x3C // LL_GAMMA_CONTRAST_CURVE_3
VAR8 = 15, 0x0F, 0x60 // LL_GAMMA_CONTRAST_CURVE_4
VAR8 = 15, 0x10, 0x7D // LL_GAMMA_CONTRAST_CURVE_5
VAR8 = 15, 0x11, 0x95 // LL_GAMMA_CONTRAST_CURVE_6
VAR8 = 15, 0x12, 0xA8 // LL_GAMMA_CONTRAST_CURVE_7
VAR8 = 15, 0x13, 0xB7 // LL_GAMMA_CONTRAST_CURVE_8
VAR8 = 15, 0x14, 0xC3 // LL_GAMMA_CONTRAST_CURVE_9
VAR8 = 15, 0x15, 0xCD // LL_GAMMA_CONTRAST_CURVE_10
VAR8 = 15, 0x16, 0xD6 // LL_GAMMA_CONTRAST_CURVE_11
VAR8 = 15, 0x17, 0xDE // LL_GAMMA_CONTRAST_CURVE_12
VAR8 = 15, 0x18, 0xE5 // LL_GAMMA_CONTRAST_CURVE_13
VAR8 = 15, 0x19, 0xEB // LL_GAMMA_CONTRAST_CURVE_14
VAR8 = 15, 0x1A, 0xF1 // LL_GAMMA_CONTRAST_CURVE_15
VAR8 = 15, 0x1B, 0xF6 // LL_GAMMA_CONTRAST_CURVE_16
VAR8 = 15, 0x1C, 0xFB // LL_GAMMA_CONTRAST_CURVE_17
VAR8 = 15, 0x1D, 0xFF // LL_GAMMA_CONTRAST_CURVE_18
VAR8 = 1, 0x00, 0x06 // SEQ_CMD

[Gamma Correction]

// Contrast Settings

VAR8 = 15, 0x000B, 0x00 // LL_GAMMA_CONTRAST_CURVE_0
VAR8 = 15, 0x000C, 0x1B // LL_GAMMA_CONTRAST_CURVE_1
VAR8 = 15, 0x000D, 0x2A // LL_GAMMA_CONTRAST_CURVE_2
VAR8 = 15, 0x000E, 0x3E // LL_GAMMA_CONTRAST_CURVE_3
VAR8 = 15, 0x000F, 0x5A // LL_GAMMA_CONTRAST_CURVE_4
VAR8 = 15, 0x0010, 0x70 // LL_GAMMA_CONTRAST_CURVE_5
VAR8 = 15, 0x0011, 0x81 // LL_GAMMA_CONTRAST_CURVE_6
VAR8 = 15, 0x0012, 0x90 // LL_GAMMA_CONTRAST_CURVE_7
VAR8 = 15, 0x0013, 0x9E // LL_GAMMA_CONTRAST_CURVE_8
VAR8 = 15, 0x0014, 0xAB // LL_GAMMA_CONTRAST_CURVE_9
VAR8 = 15, 0x0015, 0xB6 // LL_GAMMA_CONTRAST_CURVE_10
VAR8 = 15, 0x0016, 0xC1 // LL_GAMMA_CONTRAST_CURVE_11
VAR8 = 15, 0x0017, 0xCB // LL_GAMMA_CONTRAST_CURVE_12
VAR8 = 15, 0x0018, 0xD5 // LL_GAMMA_CONTRAST_CURVE_13
VAR8 = 15, 0x0019, 0xDE // LL_GAMMA_CONTRAST_CURVE_14
VAR8 = 15, 0x001A, 0xE7 // LL_GAMMA_CONTRAST_CURVE_15
VAR8 = 15, 0x001B, 0xEF // LL_GAMMA_CONTRAST_CURVE_16
VAR8 = 15, 0x001C, 0xF7 // LL_GAMMA_CONTRAST_CURVE_17
VAR8 = 15, 0x001D, 0xFF // LL_GAMMA_CONTRAST_CURVE_18

// Neutral Settings

VAR8 = 15, 0x001E, 0x00 // LL_GAMMA_NEUTRAL_CURVE_0
VAR8 = 15, 0x001F, 0x1B // LL_GAMMA_NEUTRAL_CURVE_1
VAR8 = 15, 0x0020, 0x2A // LL_GAMMA_NEUTRAL_CURVE_2
VAR8 = 15, 0x0021, 0x3E // LL_GAMMA_NEUTRAL_CURVE_3
VAR8 = 15, 0x0022, 0x5A // LL_GAMMA_NEUTRAL_CURVE_4
VAR8 = 15, 0x0023, 0x70 // LL_GAMMA_NEUTRAL_CURVE_5
VAR8 = 15, 0x0024, 0x81 // LL_GAMMA_NEUTRAL_CURVE_6
VAR8 = 15, 0x0025, 0x90 // LL_GAMMA_NEUTRAL_CURVE_7
VAR8 = 15, 0x0026, 0x9E // LL_GAMMA_NEUTRAL_CURVE_8
VAR8 = 15, 0x0027, 0xAB // LL_GAMMA_NEUTRAL_CURVE_9
VAR8 = 15, 0x0028, 0xB6 // LL_GAMMA_NEUTRAL_CURVE_10
VAR8 = 15, 0x0029, 0xC1 // LL_GAMMA_NEUTRAL_CURVE_11
VAR8 = 15, 0x002A, 0xCB // LL_GAMMA_NEUTRAL_CURVE_12
VAR8 = 15, 0x002B, 0xD5 // LL_GAMMA_NEUTRAL_CURVE_13
VAR8 = 15, 0x002C, 0xDE // LL_GAMMA_NEUTRAL_CURVE_14
VAR8 = 15, 0x002D, 0xE7 // LL_GAMMA_NEUTRAL_CURVE_15
VAR8 = 15, 0x002E, 0xEF // LL_GAMMA_NEUTRAL_CURVE_16
VAR8 = 15, 0x002F, 0xF7 // LL_GAMMA_NEUTRAL_CURVE_17
VAR8 = 15, 0x0030, 0xFF // LL_GAMMA_NEUTRAL_CURVE_18

// Noise Reduction Settings

VAR8 = 15, 0x0031, 0x00 // LL_GAMMA_NRCURVE_0
VAR8 = 15, 0x0032, 0x0D // LL_GAMMA_NRCURVE_1
VAR8 = 15, 0x0033, 0x19 // LL_GAMMA_NRCURVE_2
VAR8 = 15, 0x0034, 0x30 // LL_GAMMA_NRCURVE_3
VAR8 = 15, 0x0035, 0x56 // LL_GAMMA_NRCURVE_4
VAR8 = 15, 0x0036, 0x70 // LL_GAMMA_NRCURVE_5
VAR8 = 15, 0x0037, 0x81 // LL_GAMMA_NRCURVE_6
VAR8 = 15, 0x0038, 0x90 // LL_GAMMA_NRCURVE_7
VAR8 = 15, 0x0039, 0x9E // LL_GAMMA_NRCURVE_8
VAR8 = 15, 0x003A, 0xAB // LL_GAMMA_NRCURVE_9
VAR8 = 15, 0x003B, 0xB6 // LL_GAMMA_NRCURVE_10

```
VAR8 = 15, 0x003C, 0xC1 // LL_GAMMA_NRCURVE_11
VAR8 = 15, 0x003D, 0xCB // LL_GAMMA_NRCURVE_12
VAR8 = 15, 0x003E, 0xD5 // LL_GAMMA_NRCURVE_13
VAR8 = 15, 0x003F, 0xDE // LL_GAMMA_NRCURVE_14
VAR8 = 15, 0x0040, 0xE7 // LL_GAMMA_NRCURVE_15
VAR8 = 15, 0x0041, 0xEF // LL_GAMMA_NRCURVE_16
VAR8 = 15, 0x0042, 0xF7 // LL_GAMMA_NRCURVE_17
VAR8 = 15, 0x0043, 0xFF // LL_GAMMA_NRCURVE_18
```

Development Tool Overview

Figure 57 shows the general development flow for the MT9T111. The MT9T111 provides the following development software and hardware tools:

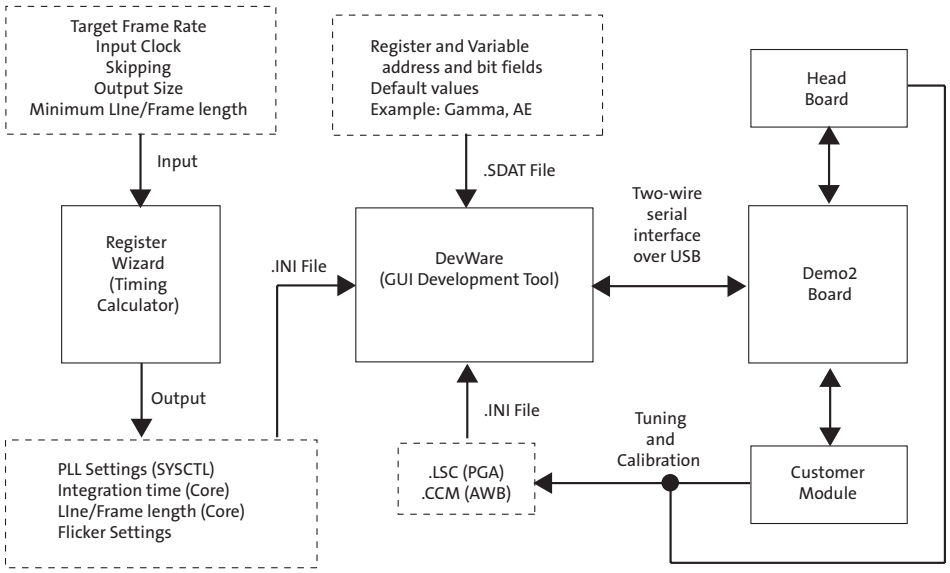
- DevWare: PC-based GUI development tool
- Register Wizard: Initial timing settings and calculator
- Demo2 board: Common demo board platform to interface between PC and various sensor boards including customer boards
- Head board: Sensor board (MT9T111)

There are three major steps for calibration and tuning for the MT9T111.

1. Timing calculation: The user can provide the clock, frame rate, output size, and so forth. to the Register Wizard. Register Wizard generates initial power-on file for powering-up the sensor.
2. Len Shading Correction (LC): The user can obtain the PGA coefficients for the particular lens. DevWare provides easy lens calibration through a GUI.
3. Color tuning: The user can also obtain CCM (color correction matrix) and tune for auto white balance using DevWare.

The following sections provide more details on each of these three calibration and tuning processes.

Figure 57: Development Tool Overview



Register Wizard

The register wizard is a software program that allows a user to generate the proper settings in regards to timing and the PLL. 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 that can easily be loaded in the Aptina's DevWare demo software.

Procedure for Generating Frame Timing Setting

After opening the MT9T111 Register Wizard tool, go to the PLL settings section to specify the input clock and target system frequencies. See Figure 58 on page 117 for a typical Register Wizard menu.

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, the user can leave the “Use Min Freq” box checked and let the tool to select the minimum PLL output frequency needed to output the frame rate indicated in the "Image Timing" section.

Note: The MT9T111 can work in parallel output or MIPI output mode. Different modes have different PLL timing settings. For the system that works in parallel mode, the checkbox **For Parallel Output Mode** must be checked. For a system running in MIPI mode only, the checkbox **For Parallel Output Mode** must be unchecked. The user also can input specific **M** and **N** values to make the checkbox **Specify M value** and **Specify N value** checked. If user inputs incorrect values for **M** or **N**, the box **Target VCO frequency (MHz)** will become red, indicating that the values must be changed.

The text window on the right will output the required PLL settings—set by the M and N values—to achieve the named configurations.

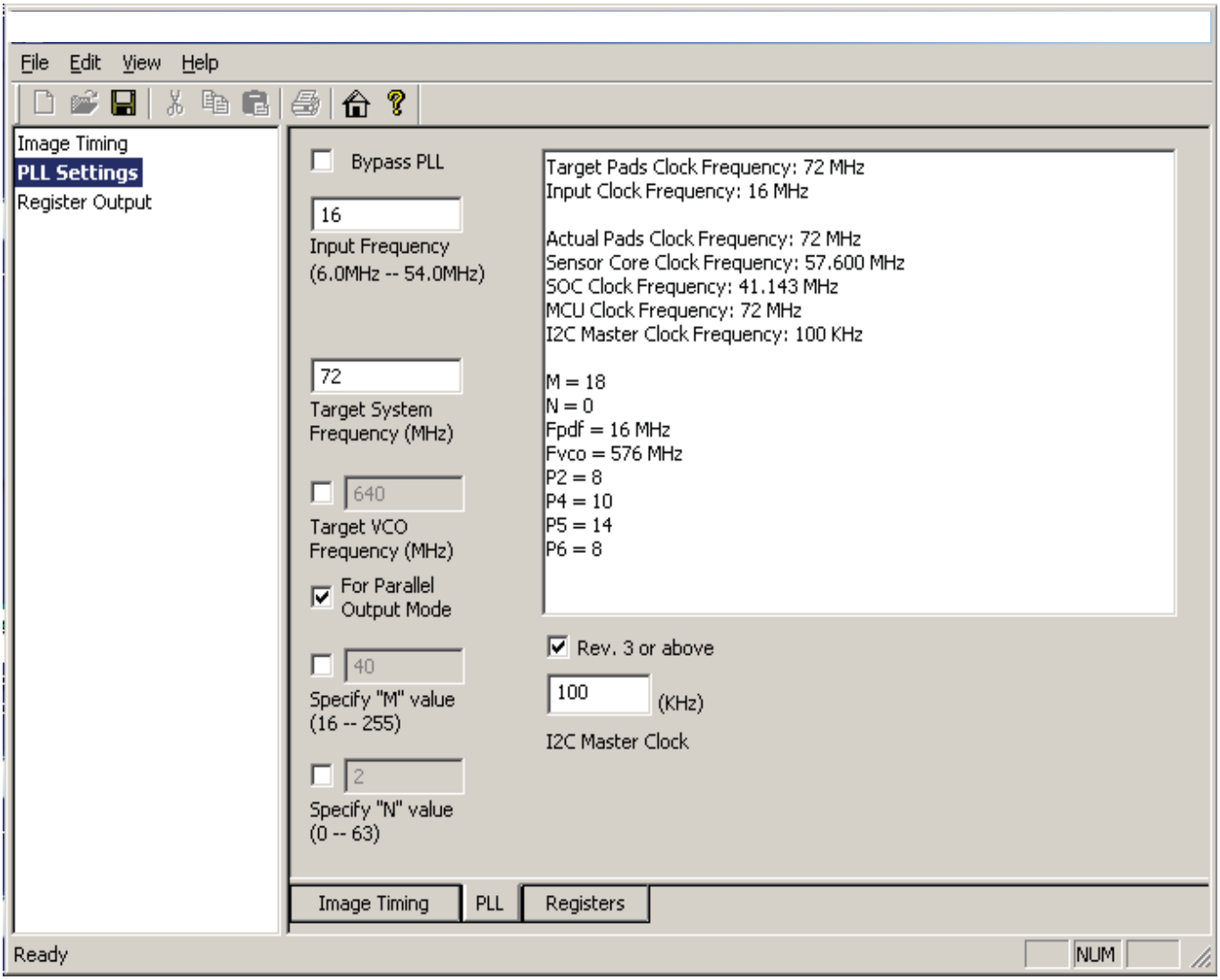
Input Clock and PLL Output Frequencies

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 384 MHz to 768 MHz.

Image Timing

The image timing parameters depend on the pixel clock frequency. Configure the clock frequency on the **PLL Settings** panel before configuring settings on this tab. Once the clock frequency is configured, the user can use this tab to configure the frame rates, output resolution, binning option, skipping option, and blanking options 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 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. It is necessary to match the line (row) time for both contexts in order for the firmware drivers (such as AE) to function optimally.

Figure 58: Register Wizard PLL Menu



Register Wizard – Register Output

To save these settings, click the diskette icon in the tool bar. Once the .ini file is generated, it may be loaded into DevWare. If the checkbox "**Output as registers" instead of "variables"** is checked, the saved .ini file with register operation format is saved, and can be transferred to a third party through the two-wire serial bus master's command set.

This file was generated by: MT9T111 Register Wizard

; Version: 2.9.0.0 Build Date: mm/dd/2007

;
; [PLL PARAMETERS]
; Bypass PLL: Unchecked
; Input Frequency: 16.000
; Use Min Freq.: Checked
; Target VCO Frequency: Unspecified

```
; For Parallel Output: Checked
; "M" Value: Unspecified
; "N" Value: Unspecified
;
; Target Pads Clock Frequency: 72 MHz
; Input Clock Frequency: 16 MHz
;
; Actual Pads Clock Frequency: 72 MHz
; Sensor Core Clock Frequency: 64 MHz
;    (using minimum based on image timing requirements)
; SOC Clock Frequency: 48 MHz
; MCU Clock Frequency: 72 MHz
;
; M = 18
; N = 0
; Fpdf = 16 MHz
; Fvco = 576 MHz
; P2 = 8
; P4 = 9
; P5 = 12
; P6 = 8
;
; [CONTEXT A PARAMETERS]
;
; Requested Frames Per Second: 27.500
; Output Columns: 1024
; Output Rows: 768
; JPEG: Unchecked
; Use Binning: Checked
; X-only Binning: Unchecked
; Allow Skipping: Unchecked
; Use Context B Line Time: Unchecked

; Low Power: Unchecked
; Blanking Computation: HB Min then VB
;
; Max Frame Time: 36.3636 msec
; Max Frame Clocks: 2327272.7 clocks (64 MHz)
; Pixel Clock: divided by 1
; Skip Mode: 2x cols, 2x rows, Bin Mode: Yes
; Horiz clks:  1032 active + 1478 blank = 2510 total
; Vert  rows:  776 active + 151 blank = 927 total
; Extra Delay: 502 clocks
;
; Actual Frame Clocks: 2327272 clocks
; Row Time: 39.219 usec / 2510 clocks
; Frame time: 36.363625 msec
; Frames per Sec: 27.500 fps
;
; 50Hz Flicker Period: 254.98 lines
; 60Hz Flicker Period: 212.48 lines
; RX Trigger Mark: 20
; OB Trigger Mark: 536
```

```
;
; [CONTEXT B PARAMETERS]
;
; Requested Frames Per Second: 10.450
; Output Columns: 2048
; Output Rows: 1536
; JPEG: Unchecked
; Use Binning: Unchecked
; X-only Binning: Unchecked
; Allow Skipping: Unchecked
; Use Context A Line Time: Unchecked
; Low Power: Unchecked
; Blanking Computation: HB Min then VB
;
; Max Frame Time: 95.6938 msec
; Max Frame Clocks: 6124401.9 clocks (64 MHz)
; Pixel Clock: divided by 1
; Skip Mode: 1x cols, 1x rows, Bin Mode: No
; Horiz clks:  2056 active + 1683 blank = 3739 total
; Vert  rows:  1544 active + 93 blank = 1637 total
; Extra Delay: 3658 clocks
;
; Actual Frame Clocks: 6124401 clocks
; Row Time: 58.422 usec / 3739 clocks
; Frame time: 95.693766 msec
; Frames per Sec: 10.450 fps
;
; 50Hz Flicker Period: 171.17 lines
; 60Hz Flicker Period: 142.64 lines
; RX Trigger Mark: 20
; OB Trigger Mark: 366
;

[MT9T111 (SOC3130) Register Wizard Defaults]
BITFIELD = 0x14, 1, 1        //   Bypass PLL
BITFIELD = 0X14, 2, 0        //  Power-down PLL
REG = 0x0014, 0x2145        //  PLL control: BYPASS PLL = 8517
REG = 0x0010, 0x0012        //  PLL Dividers = 18
REG = 0x0012, 0x0070        //  PLL P Dividers = 112
REG = 0x002A, 0x77B8        //  PLL P Dividers 4-5-6 = 30648
REG = 0x0014, 0x2545        //  PLL control: TEST_BYPASS on = 9541
REG = 0x0014, 0x2547        //  PLL control: PLL_ENABLE on = 9543
REG = 0x0014, 0x3447        //  PLL control: SEL_LOCK_DET on = 13383

POLL_REG=0x0014,0x8000 == 0x0000, DELAY=10, TIMEOUT = 5000

REG = 0x0014, 0x3047        //  PLL control: TEST_BYPASS off = 12359
REG = 0x0014, 0x3046        //  PLL control: PLL_BYPASS off = 12358
REG = 0x001E, 0x0707        //  Pad Slew Rate = 1799
REG = 0x98E, 0x6800        //  Output Width (A)
REG = 0x990, 0x0400        //       = 1024
REG = 0x98E, 0x6802        //  Output Height (A)
REG = 0x990, 0x0300        //       = 768
```

```
REG = 0x98E, 0xE88E       //  JPEG (A)
REG = 0x990, 0x00       // = 0
REG = 0x98E, 0x68A0       //  Adaptive Output Clock (A)

BITFIELD = 0x990, 0x0040, 0x0000       // = 0

REG = 0x98E, 0x4802       //  Row Start (A)
REG = 0x990, 0x000       // = 0
REG = 0x98E, 0x4804       //  Column Start (A)
REG = 0x990, 0x000       // = 0
REG = 0x98E, 0x4806       //  Row End (A)
REG = 0x990, 0x60D       // = 1549
REG = 0x98E, 0x4808       //  Column End (A)
REG = 0x990, 0x80D       // = 2061
REG = 0x98E, 0x480A       //  Row Speed (A)
REG = 0x990, 0x0111       // = 273
REG = 0x98E, 0x480C       //  Read Mode (A)
REG = 0x990, 0x046C       // = 1132
REG = 0x98E, 0x480F       //  Fine Correction (A)
REG = 0x990, 0x00C0       // = 192
REG = 0x98E, 0x4811       //  Fine IT Min (A)
REG = 0x990, 0x0375       // = 885
REG = 0x98E, 0x4813       //  Fine IT Max Margin (A)
REG = 0x990, 0x025B       // = 603
REG = 0x98E, 0x481D       //  Base Frame Lines (A)
REG = 0x990, 0x039F       // = 927
REG = 0x98E, 0x481F       //  Min Line Length (A)
REG = 0x990, 0x05D0       // = 1488
REG = 0x98E, 0x4825       //  Line Length (A)
REG = 0x990, 0x09CE       // = 2510
REG = 0x98E, 0x6C00       //  Output Width (B)
REG = 0x990, 0x0800       // = 2048
REG = 0x98E, 0x6C02       //  Output Height (B)
REG = 0x990, 0x0600       // = 1536
REG = 0x98E, 0xEC8E       //  JPEG (B)
REG = 0x990, 0x00       // = 0
REG = 0x98E, 0x6CA0       //  Adaptive Output Clock (B)

BITFIELD = 0x990, 0x0040, 0x0000       // = 0

REG = 0x98E, 0x484A       //  Row Start (B)
REG = 0x990, 0x004       // = 4
REG = 0x98E, 0x484C       //  Column Start (B)
REG = 0x990, 0x004       // = 4
REG = 0x98E, 0x484E       //  Row End (B)
REG = 0x990, 0x60B       // = 1547
REG = 0x98E, 0x4850       //  Column End (B)
REG = 0x990, 0x80B       // = 2059
REG = 0x98E, 0x4852       //  Row Speed (B)
REG = 0x990, 0x0111       // = 273
REG = 0x98E, 0x4854       //  Read Mode (B)
REG = 0x990, 0x0024       // = 36
REG = 0x98E, 0x4857       //  Fine Correction (B)
REG = 0x990, 0x0080       // = 128
```

```
REG = 0x98E, 0x4859      //  Fine IT Min (B)
REG = 0x990, 0x01E5      // = 485
REG = 0x98E, 0x485B      //  Fine IT Max Margin (B)

REG = 0x990, 0x010B      // = 267
REG = 0x98E, 0x4865      //  Base Frame Lines (B)
REG = 0x990, 0x0665      // = 1637
REG = 0x98E, 0x4867      //  Min Line Length (B)
REG = 0x990, 0x0378      // = 888
REG = 0x98E, 0x486D      //  Line Length (B)
REG = 0x990, 0x0E9B      // = 3739
REG = 0x98E, 0xC8A5      //  search_f1_50
REG = 0x990, 0x31      // = 49
REG = 0x98E, 0xC8A6      //  search_f2_50

REG = 0x990, 0x34      // = 52
REG = 0x98E, 0xC8A7      //  search_f1_60
REG = 0x990, 0x28      // = 40
REG = 0x98E, 0xC8A8      //  search_f2_60
REG = 0x990, 0x2B      // = 43
REG = 0x98E, 0xC844      //  period_50Hz (A)
REG = 0x990, 0xFF      // = 255
REG = 0x98E, 0xC845      //  period_60Hz (A)
REG = 0x990, 0xD4      // = 212
REG = 0x98E, 0xC88C      //  period_50Hz (B)
REG = 0x990, 0xAB      // = 171
REG = 0x98E, 0xC88D      //  period_60Hz (B)
REG = 0x990, 0x8F      // = 143
REG = 0x98E, 0x4846      //  RX FIFO Watermark (A)
REG = 0x990, 0x0014      // = 20
REG = 0x98E, 0x68AA      //  TX FIFO Watermark (A)
REG = 0x990, 0x0218      // = 536
REG = 0x98E, 0xE8AC      //  TX FIFO Manual Watermark (A)
REG = 0x990, 0x01      // = 1
REG = 0x98E, 0x488E      //  RX FIFO Watermark (B)
REG = 0x990, 0x0014      // = 20
REG = 0x98E, 0x6CAA      //  TX FIFO Watermark (B)
REG = 0x990, 0x016E      // = 366
REG = 0x98E, 0xECAC      //  TX FIFO Manual Watermark (B)
REG = 0x990, 0x01      // = 1
REG = 0x98E, 0x8400      //  Refresh Sequencer Mode
REG = 0x990, 0x06      // = 6
POLL_FIELD=SEQ_CMD, !=0, DELAY=10, TIMEOUT = 100   //  wait for command to be
processed
REG = 0x98E, 0x8400      //  Refresh Sequencer
REG = 0x990, 0x05      // = 5
POLL_FIELD=SEQ_CMD, !=0, DELAY=10, TIMEOUT = 100   //  wait for command to be
processed
```

Lens Calibration Procedure

This section describes how to calibrate and tune lens shading and generate PGA (Programmable Gain Adjustment) coefficients for the MT9T111.

Equipment

- JudgeII or SpectralightIII (preferred) lightbox
- Two diffusers with filter holder
- DevWare software

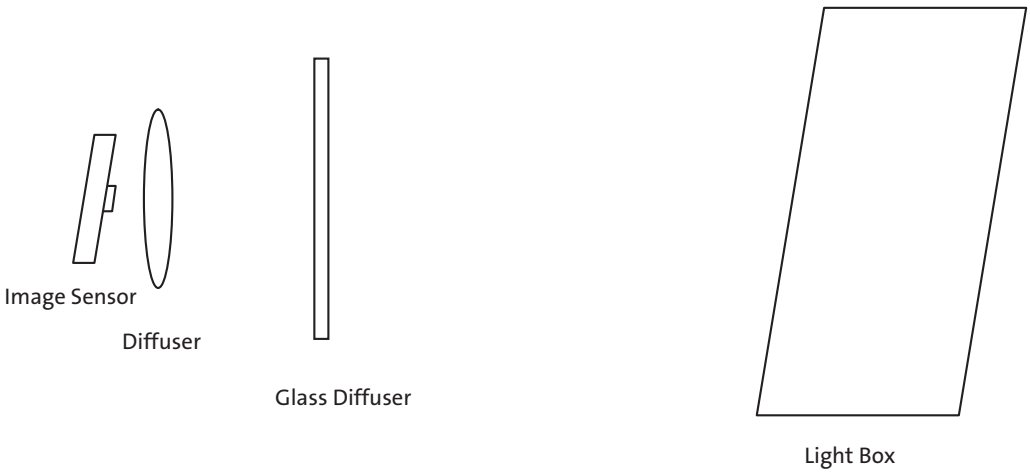
Table 36: Recommended Equipment and Settings

Equipment	Value
Common Light Sources	Illuminant A, 2850K
	TL84 Fluorescent, ~4200K
	Daylight, 6500K
Target	Uniform illumination source with de-focused lens
Lens	Matched CRA mini lens with IR cut filter
Gamma Setting	0.45, sRGB

Setup

Place one diffusion filter a few inches in front of the camera lens, place the second diffusion filter directly in front of the camera lens with the illumination source approximately 1 meter away from the glass and camera, as shown in Figure 59. The light field created from the glass should be uniform and the lens should be focused to infinity and not on the glass plate. It is important that the lens is focused as it will be used in a real camera module.

Figure 59: Lens Calibration Equipment Setup



Note: Place a black curtain over the camera system or place camera system in a dark box to prevent light reflections on the back of the diffusion glass

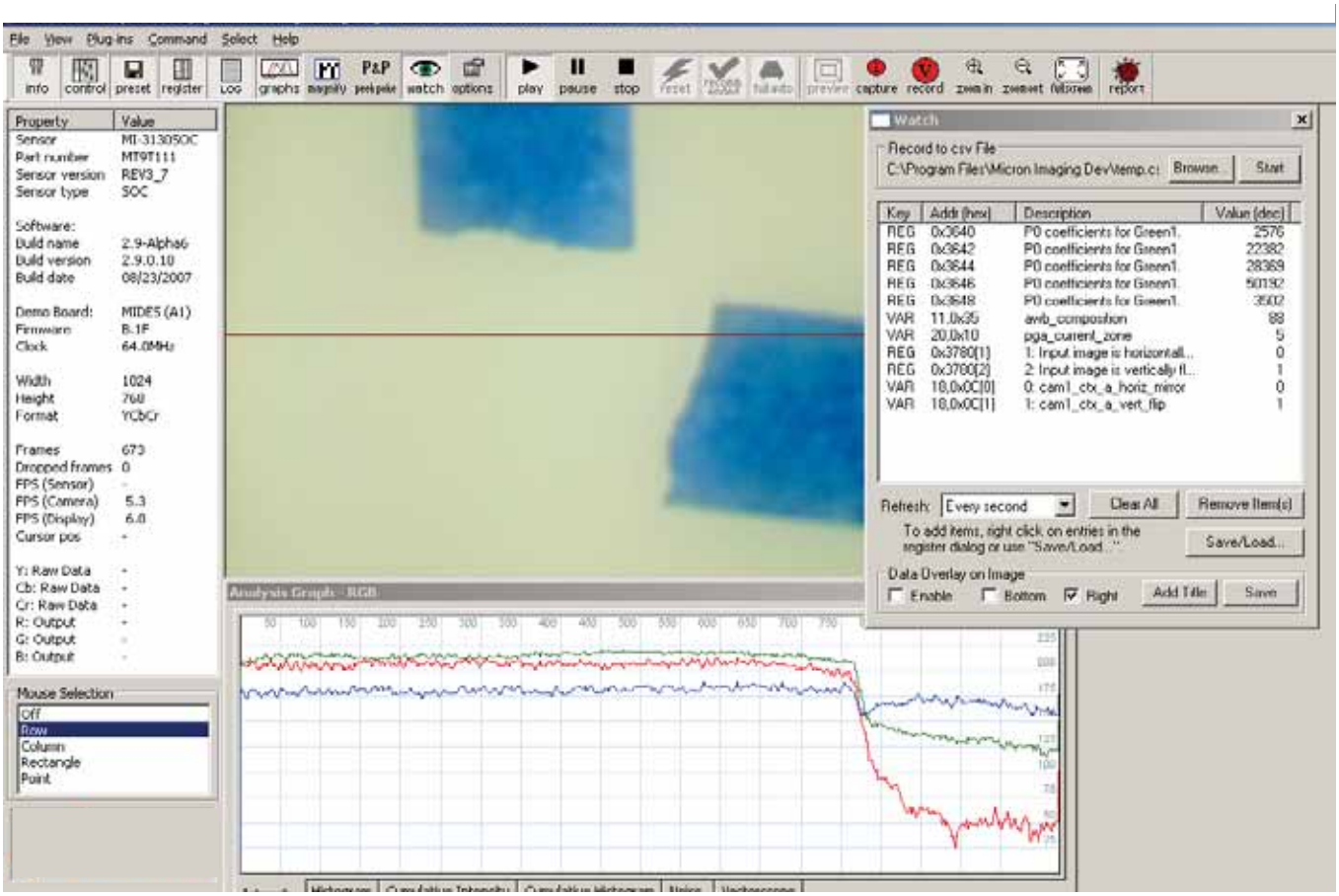
Calibration Procedure

1. Set lighting to D65.
2. Start up DevWare and camera in the default state (load default register settings).
3. Switch to full resolution.
4. Create and load a "**Lens Calibration Setup"** (See "Calibration Procedure Summary" on page 128 for more details).
   4a. Put sensor into Bypass Bayer 8+2 Mode after lens shading correction (PGA Bypass Mode).
   4b. PGA Bypass Mode should automatically bypass the following:
   – AWB, AE, flicker detection, histogram, AF drivers in the sequencer, color correction, and gamma correction
5. Set gains so that the image is white-balanced.
6. Set light level and exposure so that the brightest portion in the center of the image is about 85% of full scale with approximately gain = 1x (Be sure that no clipping occurs in any region of the image).
7. Calculate lens correction parameters.
   Option 1:
   Calculate 2nd order polynomial for X and Y directions.
   – Translate polynomial coefficients into lens correction register values.

   Option 2:
   Use DevWare tool. (See "Calibration Procedure Summary" on page 128 for more details.)
   – To find the optical center of the lens.
   – To find the optimal lens correction register values.
8. After applying lens correction, save and capture processed RGB image after lens correction is applied.
9. Change the firmware variable responsible for flipping the output image. Be sure to do a refresh mode after each change.
   VAR = 18,12,0x01,0 // 0: cam1_ctx_a_horiz_mirror
   VAR = 18,12,0x02,1 // 1: cam1_ctx_a_vert_flip
10. Check if the PGA variables responsible for flipping the applied PGA correction reflects the status of the output image.
    BITFIELD = 14208,0x0002,0 // 1: Input image is horizontally flipped.
    BITFIELD = 14208,0x0004,1 // 2: Input image is vertically flipped.

Note: You can use small pieces of painter's tape on the glass diffuser plate to detect whether or not the image was flipped. Check the image in Figure 60 on page 124 to ensure the image is display properly.

Figure 60: Check Image to See if it is Flipped Correctly

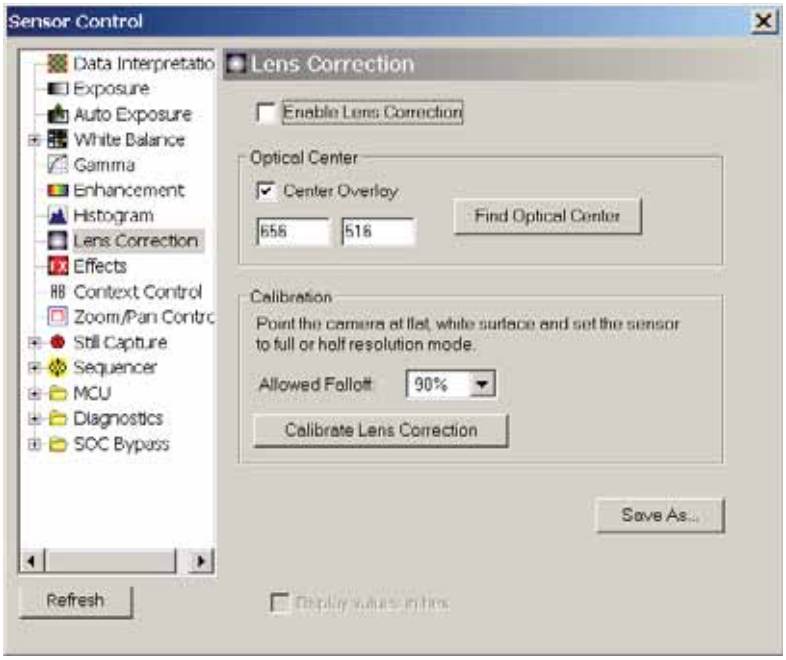


11. Save and capture processed RGB image after lens correction is applied.
12. Repeat steps 9–11 until the following combinations have been captured:
    12a. No Horizontal Flip – No Vertical Flip.
    12b. No Horizontal Flip – Vertical Flip.
    12c. Horizontal Flip – Vertical Flip.
    12d. Horizontal Flip – No Vertical Flip.
13. If required, perform the calibration procedure for the following additional illuminants. If you are trying to use adaptive PGA these additional calibrations must be performed:
    – Daylight (D65, 6500K)
    – Fluorescent (4200K)
    – Incandescent (2850K)
14. Change lighting to incandescent (A light).
15. Repeat steps 2 – 6.
16. Apply calibration settings from step 7.
17. Repeat step 8.
18. For multiple sensors repeat the sequence (steps 10 – 12) for D65 light and A light.

Using DevWare for the Lens Calibration

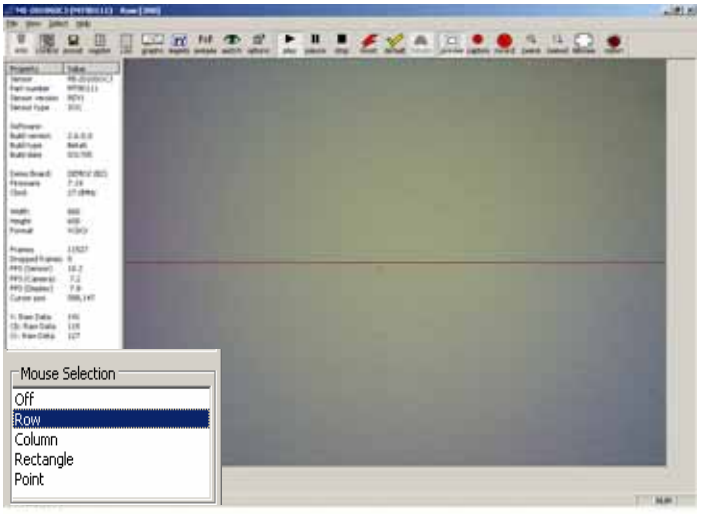
1. Find the center of the lens by going to **Sensor Control -> Lens Correction** as shown in Figure 61. Click on **Find Optical Center** and watch the zones readjust on the image.

Figure 61: Lens Regions



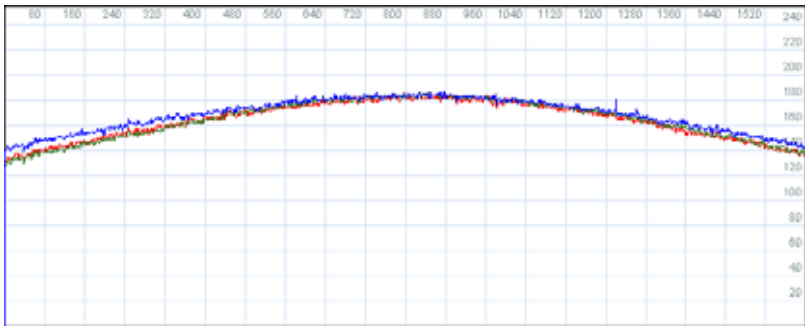
2. Set the horizontal cursor to the center of the row line in the image as shown in Figure 62. In full resolution mode, the center of the row line is set at 240 for MT9T111. The mouse cursor position can be read from the **Info** panel on the left side of the screen.

Figure 62: Sensor Array and Row Column Selection



3. Open **Analysis Graph** and view the Intensity plot of the middle row/column of the image. The graph should be similar to Figure 63.

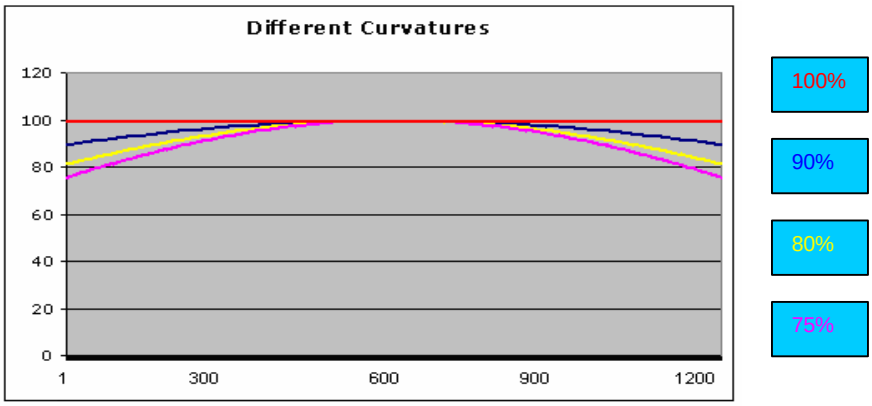
Figure 63: Lens Correction Curves



4. If the peak intensity of the green curve is not between 180—200, adjust the integration time in sensor core R0x3012 to meet this condition.
   The red and blue curve should not clip—intensity value should not be at or beyond 255—in this condition. If the clipping occurs, try using a different uniform light source or manually lower the analog gains from 0x302E (Analog Blue Gain) and 0x302C (Analog Red Gain). It is not necessary to overlap all three curves since the CCM calibration will address this issue. In addition, the lens shading correction results will not include values of analog gains.

5. Go to Sensor Control -> Lens Correction and select **Allowed Falloff**. The percentage of the lens correction can be reduced below 100 percent for the targeted effect. The user has the option to not completely calibrate the lens to obtain flat intensity curves by selecting different percentage for curvature. One hundred percent corresponds to a completely flat curve. Figure 64 shows the correlation between percent and curvature.

Figure 64: Curve Percentages



6. Push the **Calibrate lens correction button**; the lens correction calibration will finish quickly and automatically.
7. Check the **Enable Lens correction** checkbox; the new lens correction setting is active now.
8. Push the **Save as** button; the new lens correction settings will be saved in DevWare .ini file format.

Calibration Procedure Summary

1. Power the sensor.
2. Open DevWare and load the default registers.
3. **Press** the **Preview Button** to turn off the preview and go into full resolution mode. (Default is in preview mode.)
4. **Press** the **Stop Button.**
5. Load the bypass full resolution settings below:

[Bypass Full resolution]

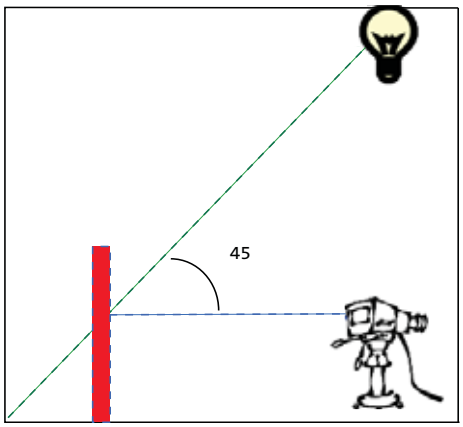
REG = 0x001C, 1281// MCU_BOOT_MODE
REG = 0x0010, 288// PLL_DIVIDERS
REG = 0x0012, 144// PLL_P_DIVIDERS
REG = 0x002A, 30719// PLL_P4_P5_P6_DIVIDERS
REG = 0x002C, 12407// PLL_P7_DIVIDER
REG = 0x0112, 532// RX_FIFO_WATERMARK
REG = 0x3C5C, 32// OB_TRIG_MARK
REG = 0x3330, 256// OUTPUT_FORMAT_TEST
REG = 0x300C, 3985// LINE_LENGTH_PCK
REG = 0x3C68, 2056// OB_LINE_PIXEL_CNT
REG = 0x3C66, 1544// OB_FRAME_LINE_CNT

6. **Press** the **Options Button** from the top down menu and enter the correct Bayer Size (2056 x 1544).
7. **Select Bayer 8 + 2**.
8. **Press Apply.**
9. **Press Play. (You are now in Bayer mode.)**
10. **Open the Sensor Control Window.**
11. **Press “Find Optical Center”.**
12. **Select Allowed Fallout.**
13. **Press “Calibration Lens Correction”**
14. **Save PGA coefficients generated by Devware to .ini file by pressing “Save As” button.**

Color Tuning Procedure

1. Setup
   1a. Hardware setup
   1b. Ensure 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 non-reflecting black tape.
   1c. Place the color rendition chart in the middle of the image screen to avoid corner color shading effect to the chart. Ensure that the entire chart is in the image screen.
   1d. Turn on one of the light sources (Daylight at 6500 Kelvin or Incandescent light at 2850 Kelvin). For the procedure described in Figure 65 on page 129, the calibration for Daylight is shown first, and then ask the user to repeat the process for incandescent light.

Figure 65: Color Tuning Lab Setup



2. Software setup
   2a. Start up DevWare and camera module.

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

3. Preset and Load

The following example code shows how to put the MT9T111 into pre-calibration mode in DevWare.

The Color correction Setup accomplishes the following:

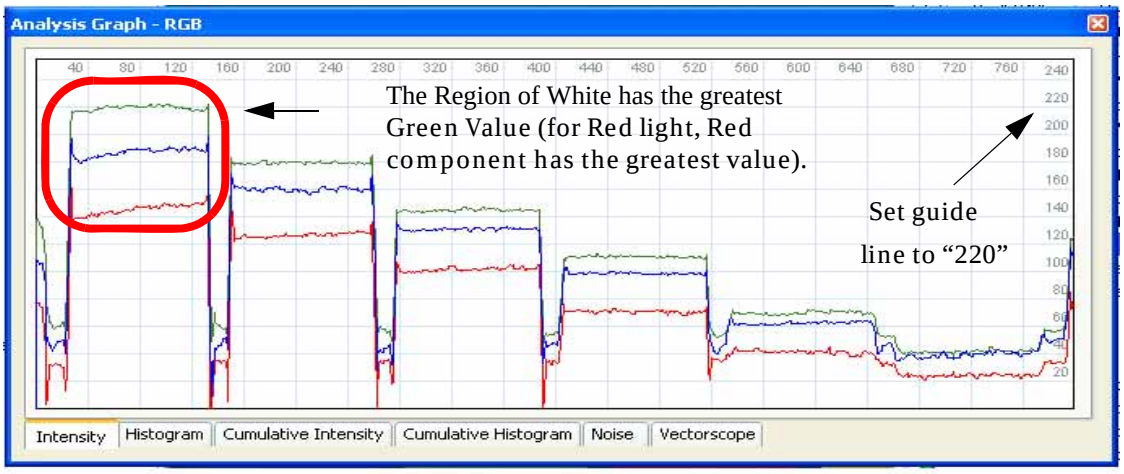
- Turns off histogram stretch
- Forces unit digital gain in AWB driver
- Turns off AWB, AE, Flicker Detection, and Histogram
- Turns off color correction and gamma correction on color pipeline

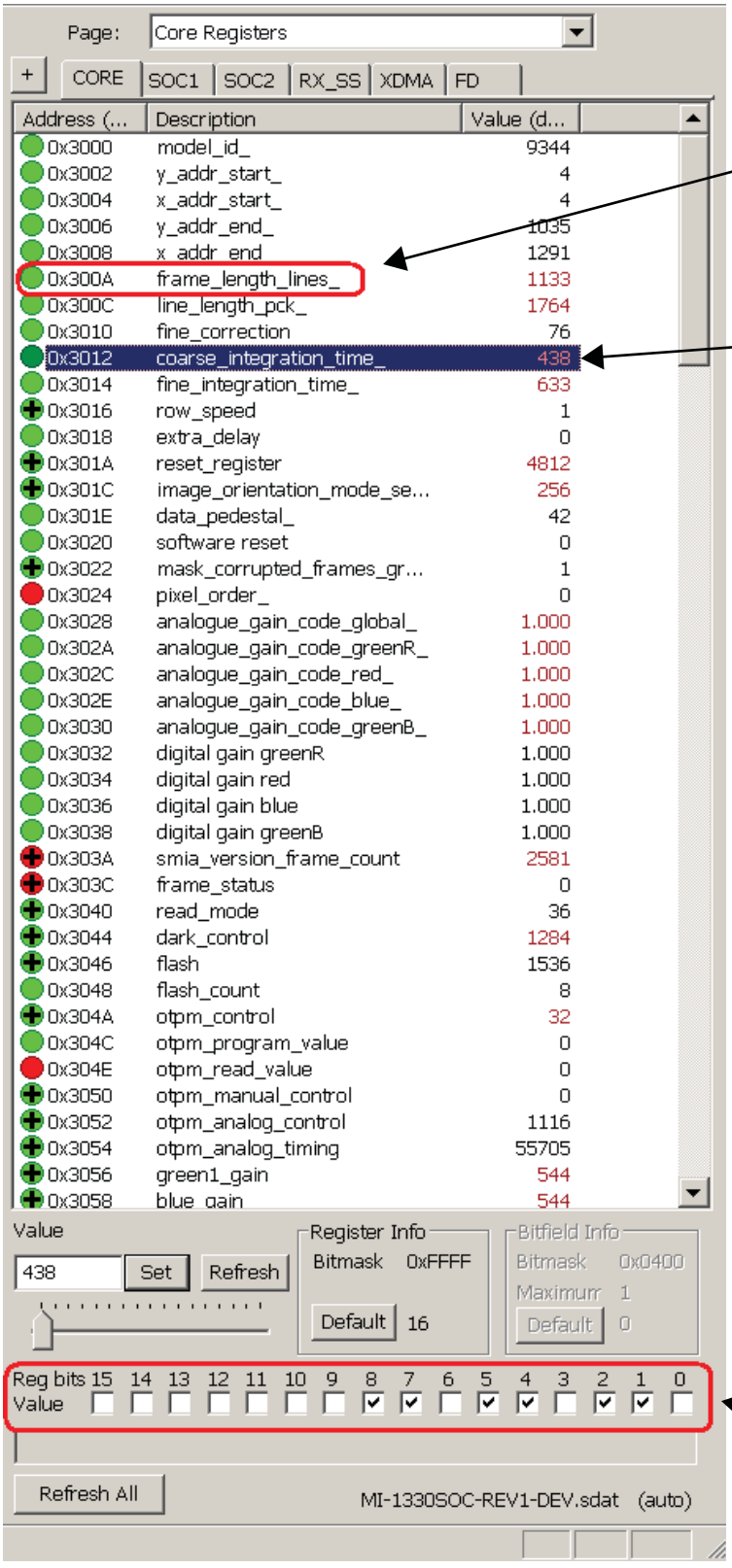
After loading color correction setup, load the settings for lens calibration. The user should calibrate the lens before calibrating color. For the procedure on lens calibration, refer to “Lens Calibration Procedure” on page 122.

4. Calibration in DevWare

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

4a. Open the **Cumulative Intensity Graph** and adjust the shutter width so the Green in white patch is at 220.



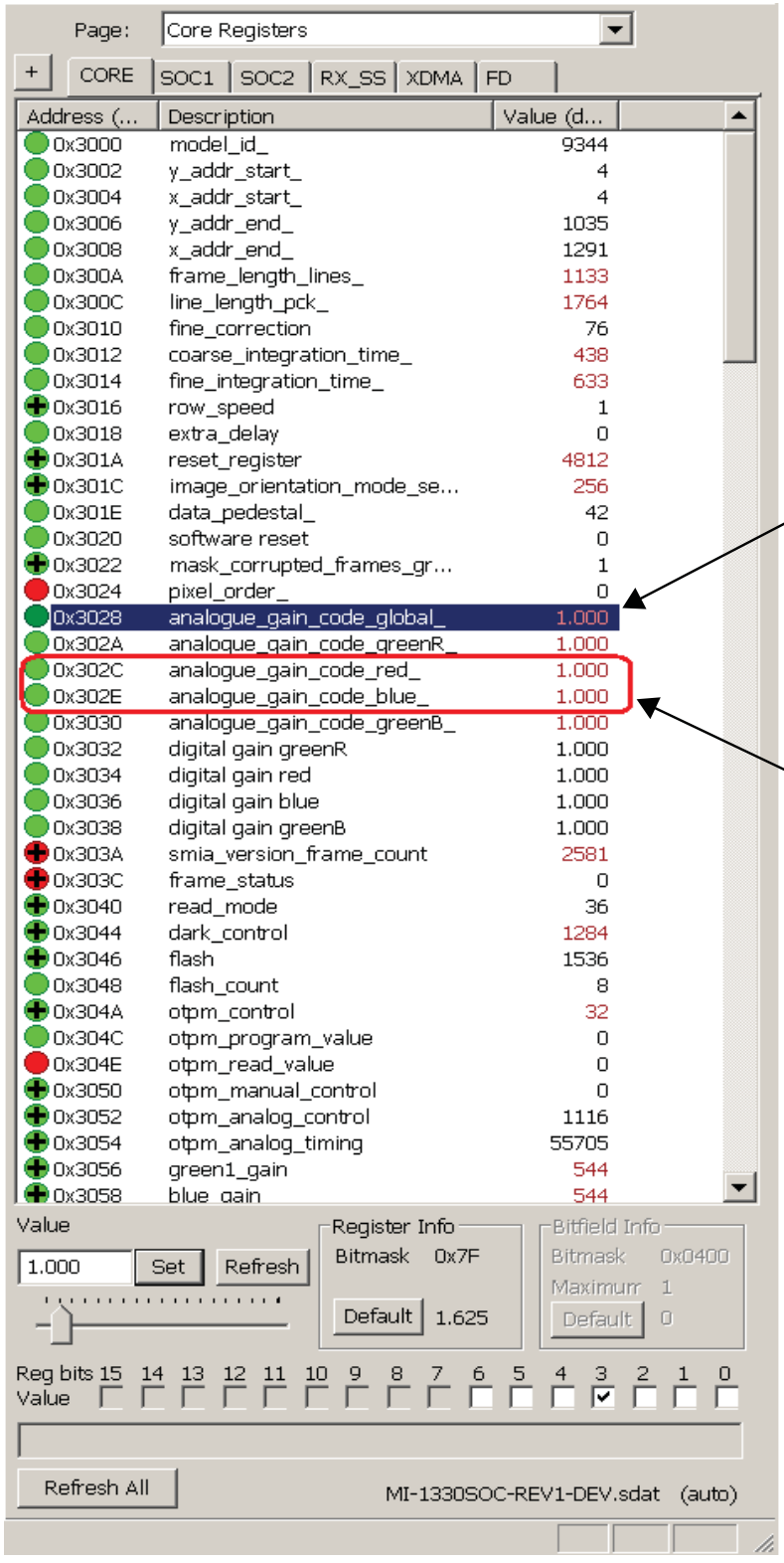


Make sure frame_length_lines > coarse_integration time.

Adjust "coarse_integration_time" register values so the Maximum Green in the Intensity graph reaches "220" as shown in the graph above.

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

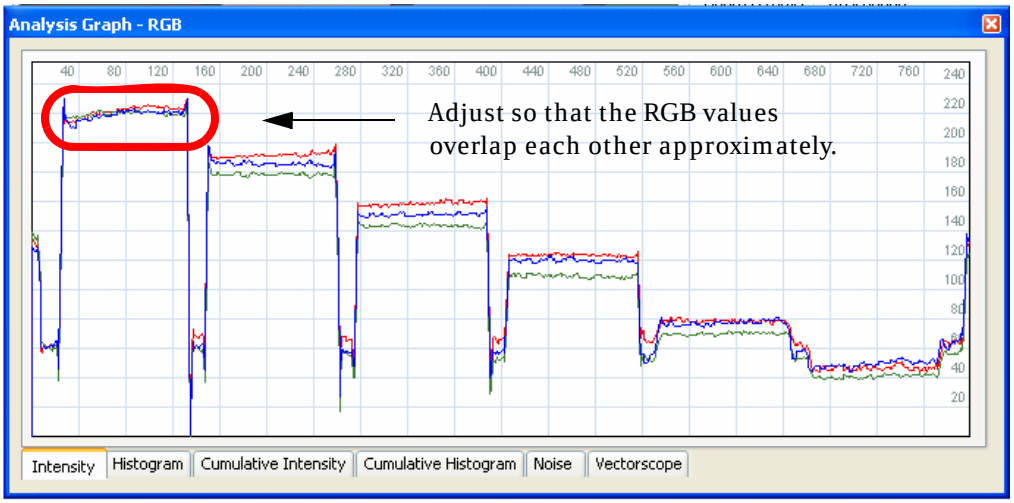
4b. Adjust the Red and Blue Gain so that the curves overlap each other approximately. Start by adjusting Red gain and then adjust Blue gain.

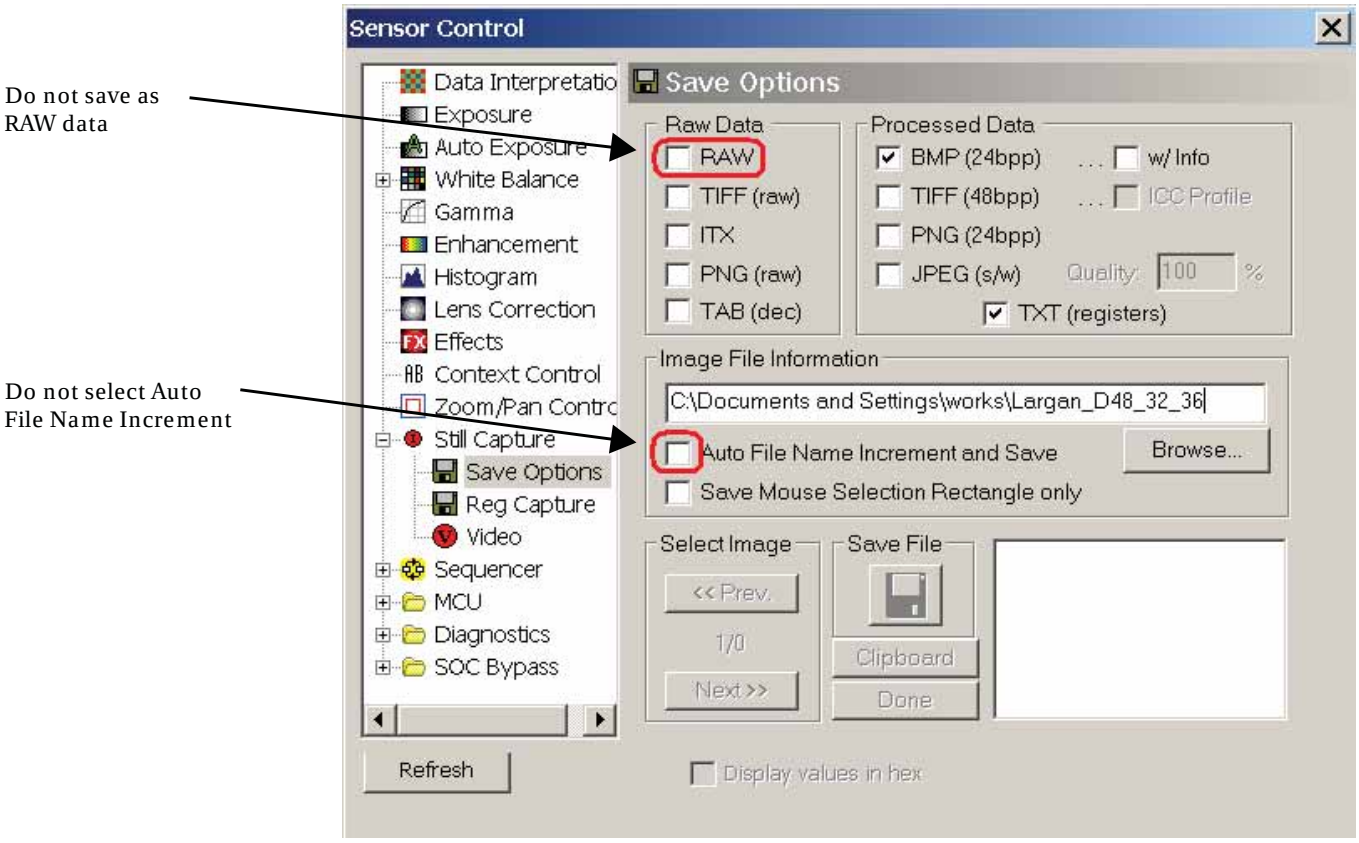


Start from adjusting Red gain and move to Blue gain (note that Green Gain should not be adjusted, it should be the default 1.000 or 0x0008).

For red light, start from adjusting blue gain. Green gain should not be adjusted (1.000 or 0x0008). If Red is higher than Green, then lower Red Gain so it overlaps with Green.

4c. Capture the Macbeth chart picture and save it as a bitmap file—the naming convention is important. 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, capturing an image taken by the Largan module with D65 and Red Gain = 48, Green Gain = 32, and Blue Gain = 36 is shown. The file name for above example will be Largan_D48_32_36. This example is used throughout the section.





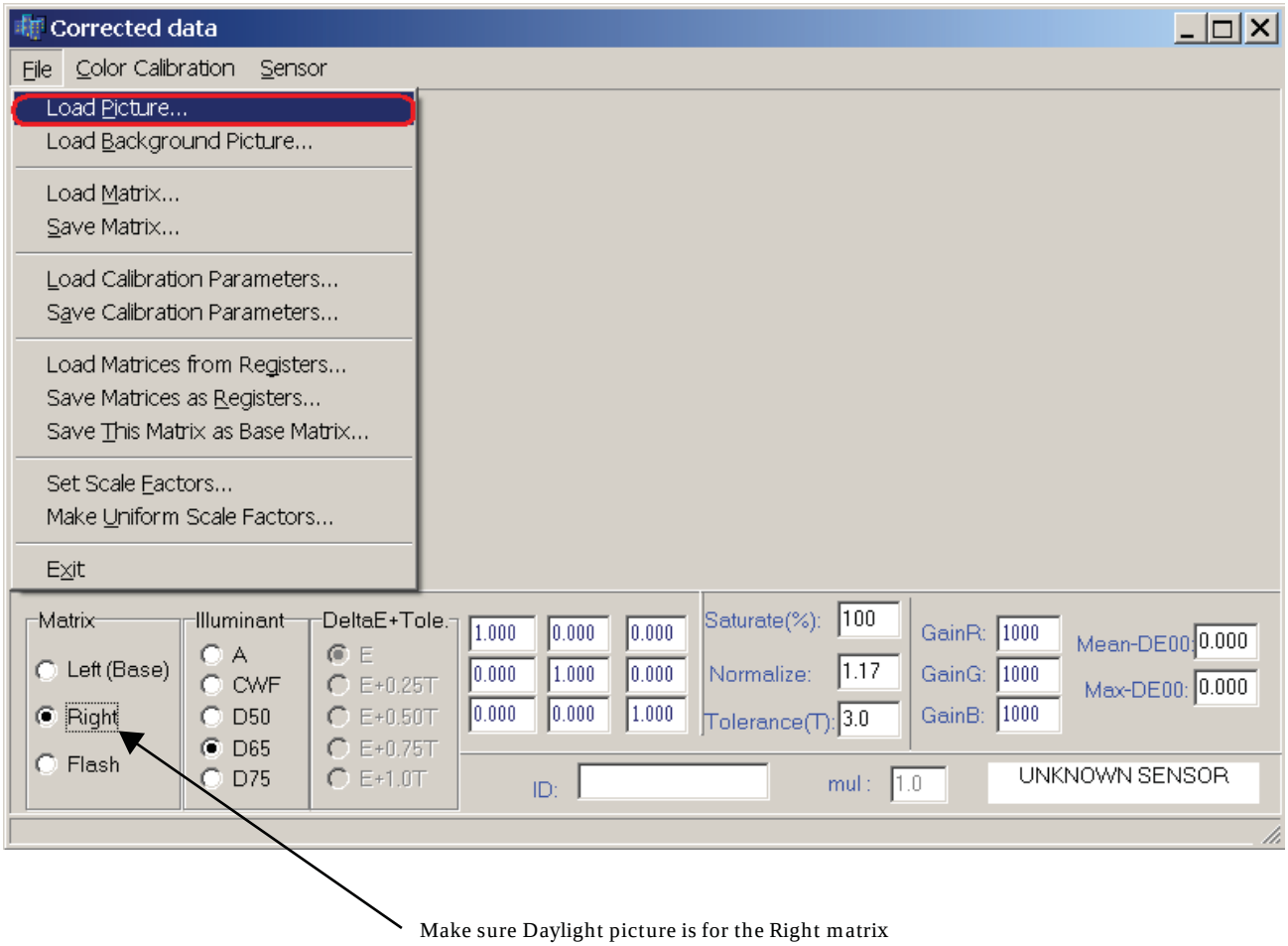
4d. With all settings staying the same, take away the Color Rendition Chart from the light box and take an image of the background. Capture this image and save it as "Largan_D48_32_36_bk." Note that "_bk" stands for background.

4e. Repeat steps 1–3 for incandescent light and save the color chart picture and the background picture as a different file name.

Note: After switching to incandescent light, reset the sensor and start from the default state.

5. Calibration of Color Correction Matrix

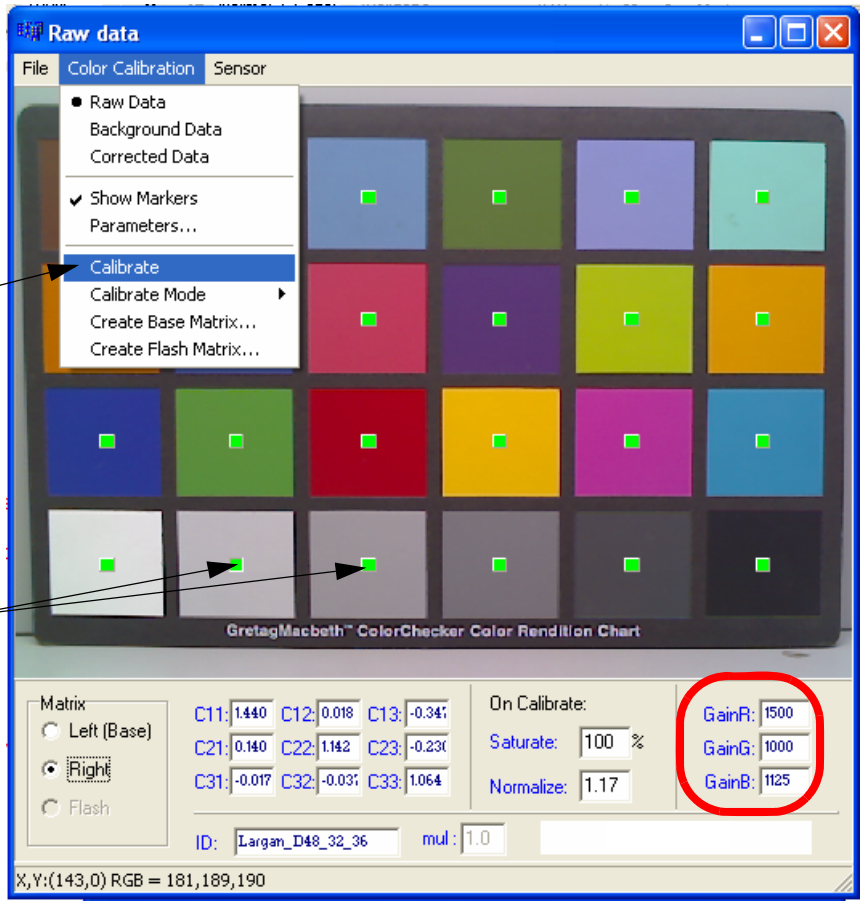
5a. Load the 'Largan_D48_32_36.bmp' as the 'Right' Picture. Note the 'Largan_D48_32_36_bk.bmp' will be loaded automatically as the background picture.



Make sure Daylight picture is for the Right matrix

5b. Manipulate and adjust so that the Matrix dots are approximately located at the middle of each color square.

Note: If the GainR is less than 1000 for Incandescent Light, change the number to 1000—then go to Color Calibration and select Calibrate.



Select Calibrate

Try to place the matrix dots at the center of the color squares

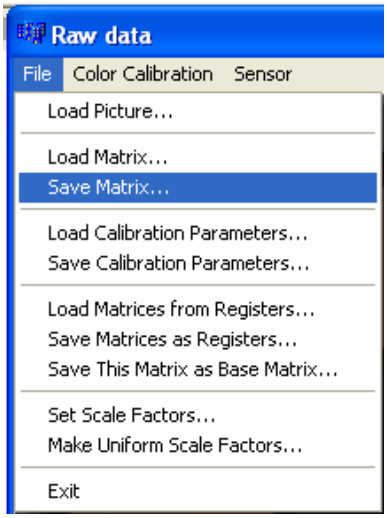
These gain ratios are automatically loaded when the picture is loaded.

GainR = 1000*(Red Gain/Green Gain)

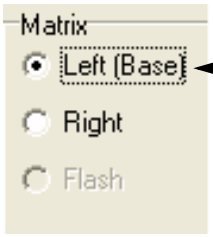
GainG = 1000*(Green Gain/Green Gain)

GainB = 1000*(Blue Gain/Green Gain)

5c. Save the calibrated Daylight picture as Matrix data file.

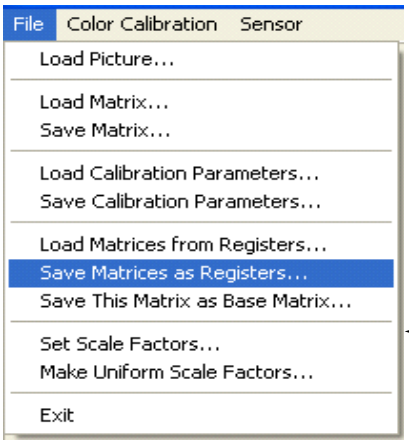


5d. After Calibrating the picture, repeat step 5a. through 5d, loading the Incandescent light picture as the 'Left (Base)' Matrix.



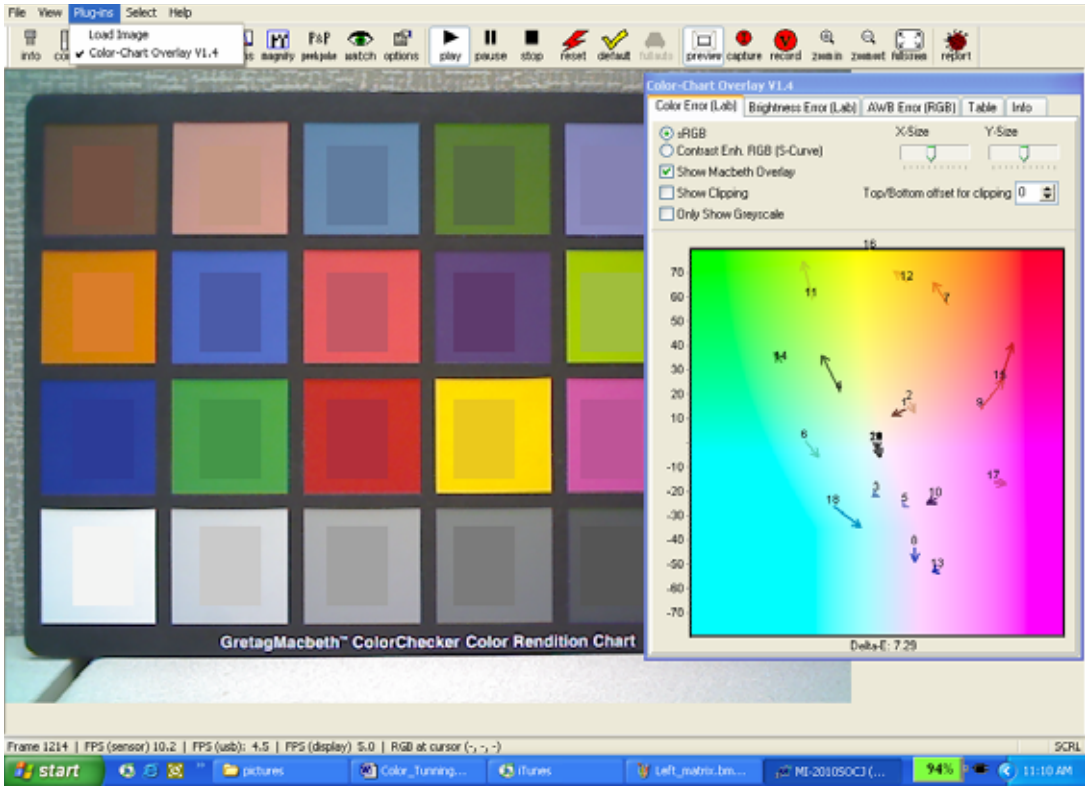
Load the Largan_D48_32_36 Picture for “Left (Base)” Matrix

5e. After both pictures have been calibrated and the corresponding matrix data file has been saved, save a combined .ini file.



This will be the .ini file which is appended to the end of the main .ini code

6. Verification
   - 6a. Hardware setup is the same as previously described.
   - 6b. Load DevWare and reset all registers (this will enable AWB, Color Correction, AE, and Gamma Correction).
   - 6c. Load the Lens Correction setting, then load new CCM setting.
   - 6d. Go to Plug-ins-> Color-char Overlay v1.4 (version may change as DevWare is updated). Ensure the patches are right on top of the color bars in the image. If not, the user can adjust X-Size and Y-Size to have the patches positioned.
   - 6e. Turn on the D65 light source or A28 light source to check the color accuracy.
   - 6f. The Delta-E parameter on Color-Chat Overlay V1.4 should be no more than 8.0.



6g. Check the value of current ccm position (AWB variable 0x0053) under different light sources. For incandescent light, the current ccm position value should be close to 0; for daylight, the current ccm position should be close to 127.

6h. Check under different light sources, the values of red gain (AWB variable 0x004E), green gain (AWB variable 0x004F), and blue gain (AWB variable 0x0050). The values should be close to 1.

6i. If needed, repeat the color correction procedure to obtain improved register setting values and repeat the verification process to check image quality.

Calibration of True Gray (TG) Limits

Note: Lens shading calibration must be performed before calibrating TG limits.

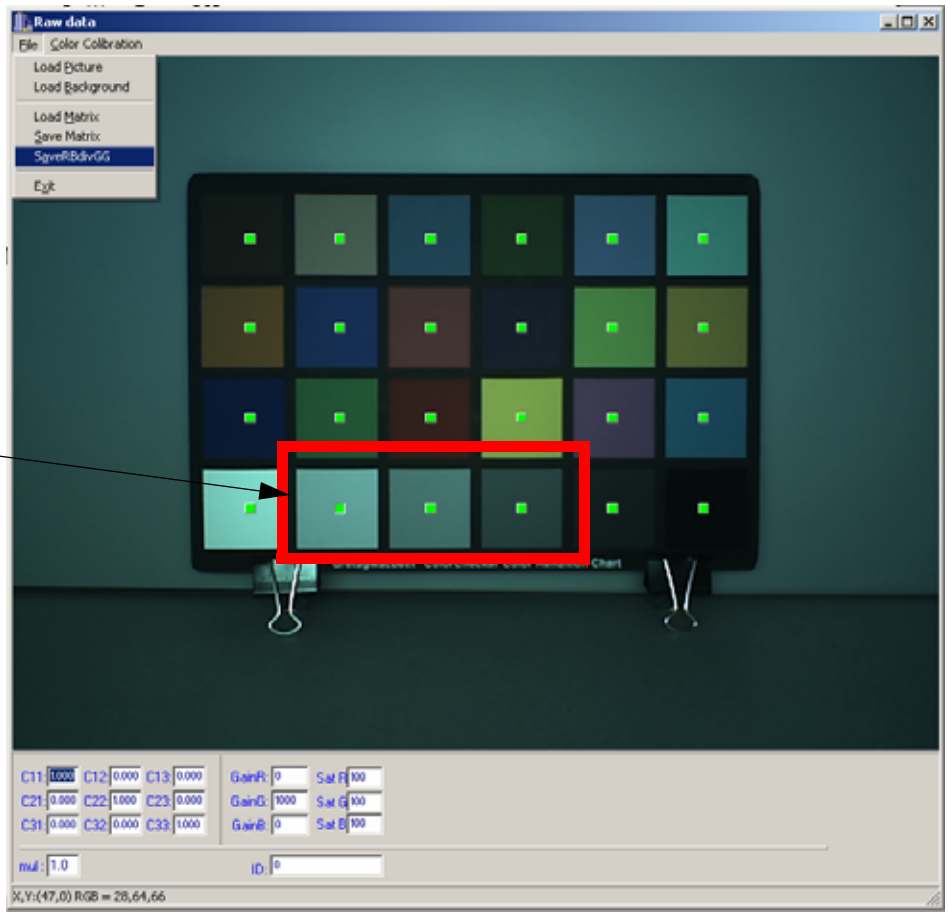
1. Setup
   The setup is the same with color calibration.
2. Preset and Load
   [True Gray AWB Calibration Setup]

The True Gray setup has done the following:

- Turned off histogram stretch
- Forced unit digital gain in AWB driver
- Turned off color correction and gamma correction on color pipeline

3. Calibration
   3a. Place and center the camera in the color temperature light box pointing at the Macbeth chart.
   3b. Load DevWare and turn on D65 light source.
   3c. Make sure sequencer variable 0x0002 = 0xF.
   3d. Load Lens shading correction settings.
   3e. Capture an image. Repeat for CWF, U30, and A28. There should be a total of 4 images captured.
   3f. Load the CCCM.exe application. Go to **File->Load Picture** and select the captured D65 image.
   3g. Enable markers by selecting **Color Calibration->Show Markers**.
   3h. Place the 24 markers on their corresponding color from the Macbeth chart.
   3i. Save RB/GG values by going to **File->SaveRBdivGG**. The results will be in a .dat file.

Record the values for these three patches from each .dat file.



3j. Repeat step 5 for the remaining three images.

3k. Record the true gray values for the gray color squares (middle 3 squares from the MacBeth color on the bottom row—patch#20, #21, #22) from each .dat file.

3l. With all 12 values (4 images x 3 squares) considered above, find the minimum and maximum values and set the max and min true gray values at the variables:

- Minimum True gray values: AWB variable 0x0063
- Maximum True gray values: AWB variable 0x0064

Note: If the values are negative, they should be set in two's complement form. A refresh command is not needed for the new TG values to be effective.

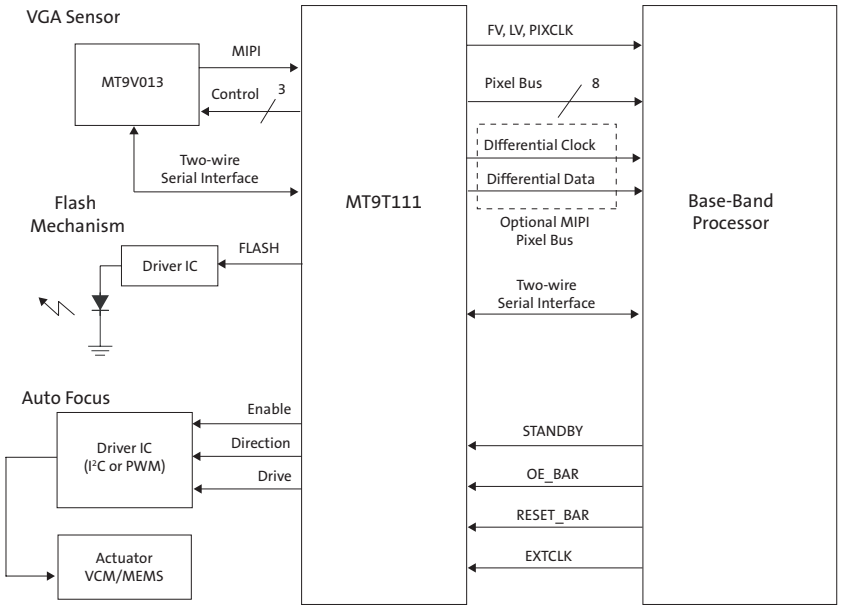
Appendix A – Dual Camera Implementation

In a typical cellular phone design, there are several submodules that can be implemented to extend the functional performance of the product. One such module is the camera subsystem, which includes a sensor, image processing, compression and interaction with the host processor. There are two typical implementations that are considered:

- Single Camera—A single image sensor is integrated and provides support for viewfinder and still capture function
- Dual Camera—A main image sensor which is typically larger than 1.3Mp used for high-quality still image capture and secondary high-quality VGA resolution image sensor.

The MT9T111 image sensor has an integrated input MIPI interface to support a simplified system design for dual-camera imaging systems, as shown in Figure 66. The figure shows the interfaces between system components (base-band processor, VGA sensor, FLASH LED, Auto Focus mechanism, and so forth) and the main camera (MT9T111).

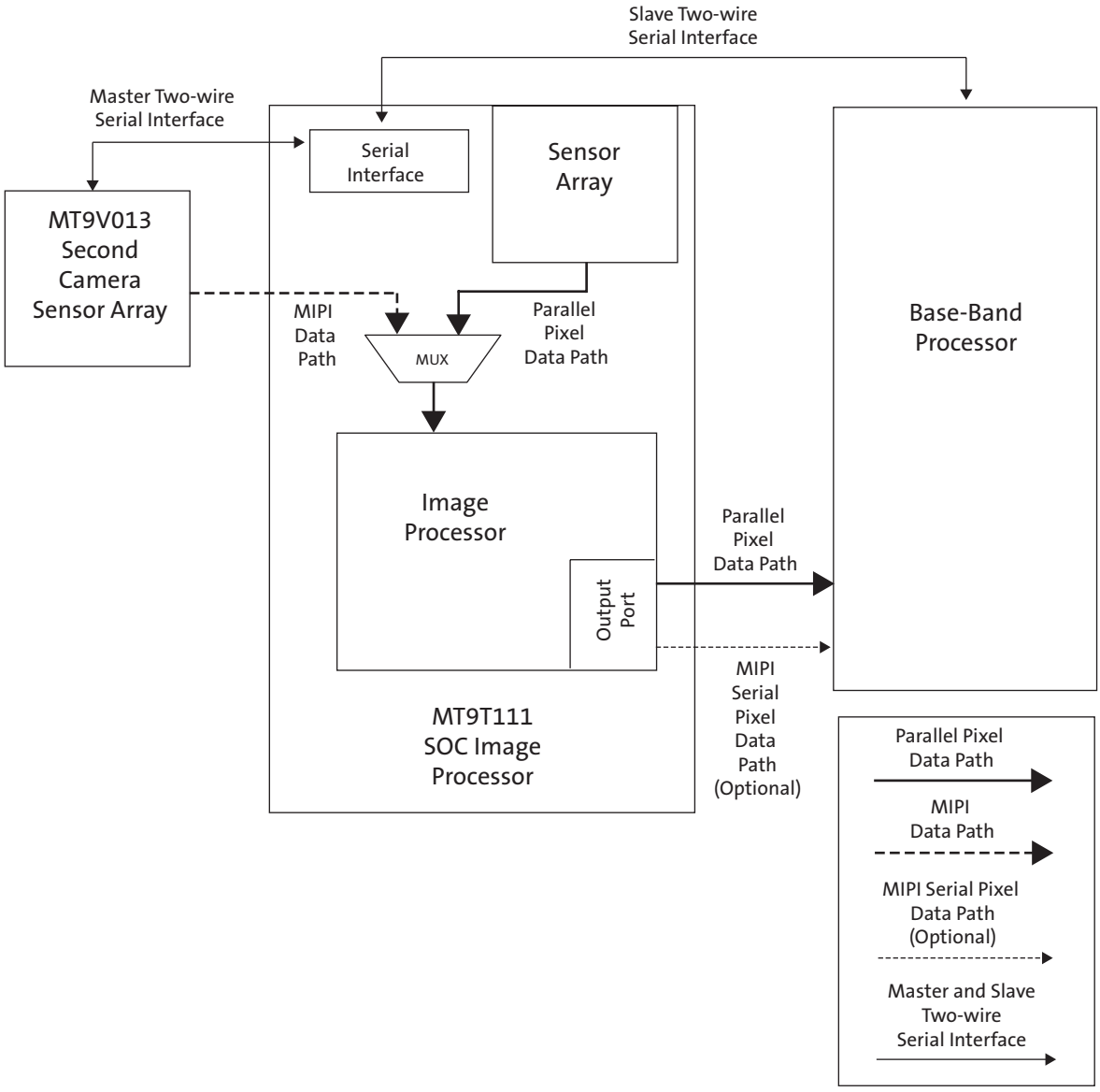
Figure 66: Dual Camera System Level



The integration of the control and pixel data bus for the MT9V013 image sensor allows the host processor to interchange seamlessly between the two image sensors without the complicated signaling protocols.

This second sensor support will allow the MT9V013 to utilize the resources of the MT9T111 image sensor. The shared image processing capabilities will allow the host processor to off-load all the image processing functionality, as shown in Figure 67.

Figure 67: Dual Camera Data Flow Diagram



A typical dual camera interconnect drawing using the MT9T111 and MT9V013 is shown in Figure on page 144.

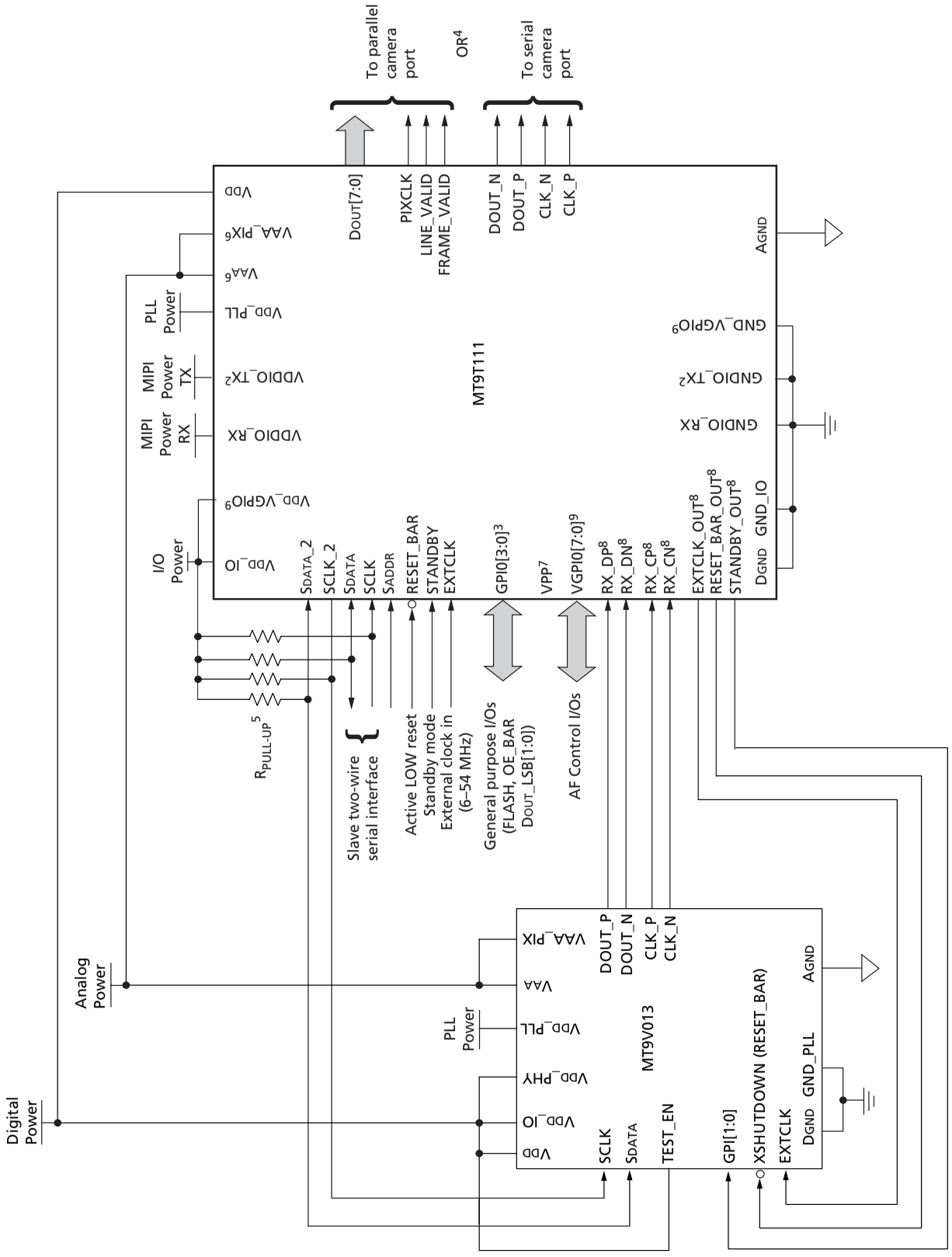


Figure 68: Dual Camera Typical Interconnect

Notes:
1. This typical configuration shows only one scenario out of multiple possible variations for this sensor.
2. If a MIPI Interface is not required, the $V_{DD}IO\_TX$ and $V_{DD}IO\_RX$ pads must be connected to V_{DD} and the GNDIO_TX and GNDIO_RX pads must be connected to D_{GND}. The following signals must be left floating: DOUT_P, DOUT_N, CLK_P, and CLK_N.
3. The GPIO pads can serve multiple features that can be reconfigured. The function and direction will vary by applications.
4. Only one of the output modes (serial or parallel) can be used at any time.
5. Aptina recommends a resistor value of 1.5KΩ to V_{DD}_IO for the two-wire serial interface $R_{PULL-UP}$; however, greater values may be used for slower transmission speed.
6. V_{AA} and VAA_PIX must be tied together.
7. V_{PP} is the one-time programmable (OTP) memory programming voltage and should be left floating during normal operation.
8. If the bridging function to the MT9V013 is not required, the following signals can be left floating: EXTCLK_OUT, RESET_BAR_OUT, STANDBY_OUT, RX_DP, RX_DN, RX_CP, and RX_CN.
9. If AF is not required, the following pads can be floating: V_{DD}_VGPIO, GND_VGPIO and VGPIO[7:0].

Appendix B – Demo Board Systems

Three boards are provided to support both parallel and MIPI output:

- Head board
- Demo2 board
- MIPI receiver board

The head board has an MT9T111 sensor and corresponding CRA lens. The parallel port signals are available through a 26-pin header, and the MIPI port signals are available through an RJ45 connector. The Demo2 board is the common platform for all Aptina

sensor head boards to interface to PC USB port. The Demo2 board has a USB controller and an FPGA to convert parallel data outputs to the USB port. The headboard and the Demo2 board communicate through a two-wire serial interface. Both boards are powered by the USB interface. Figure 69 shows the block diagram of the head board and the Demo2 board in parallel data configuration.

Figure 70 on page 146 shows the connection for using the MIPI receiver board between the head board and the Demo2 board for the MIPI interface. The MIPI receiver connects to the head board through a RJ45 connector. The MIPI receiver board converts the serial MIPI data back to parallel data and routes it to the Demo2 board. The MIPI receiver board needs a separate (+5V) power supply. The external 5V supply should be turned on before the USB connection.

Figure 69: Demo Board (Parallel Mode)

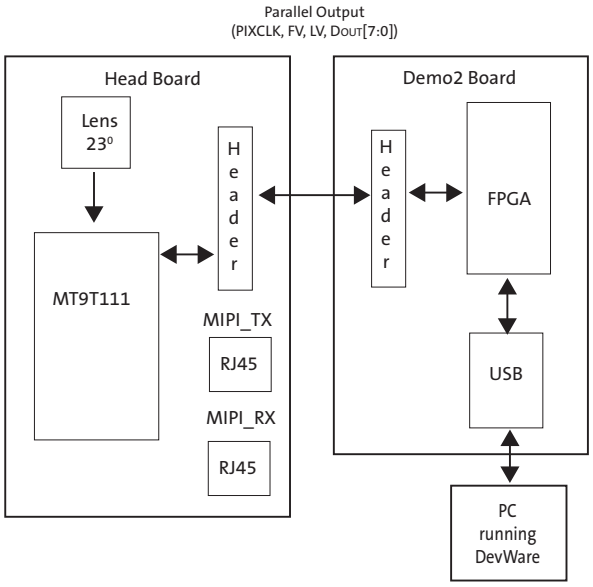
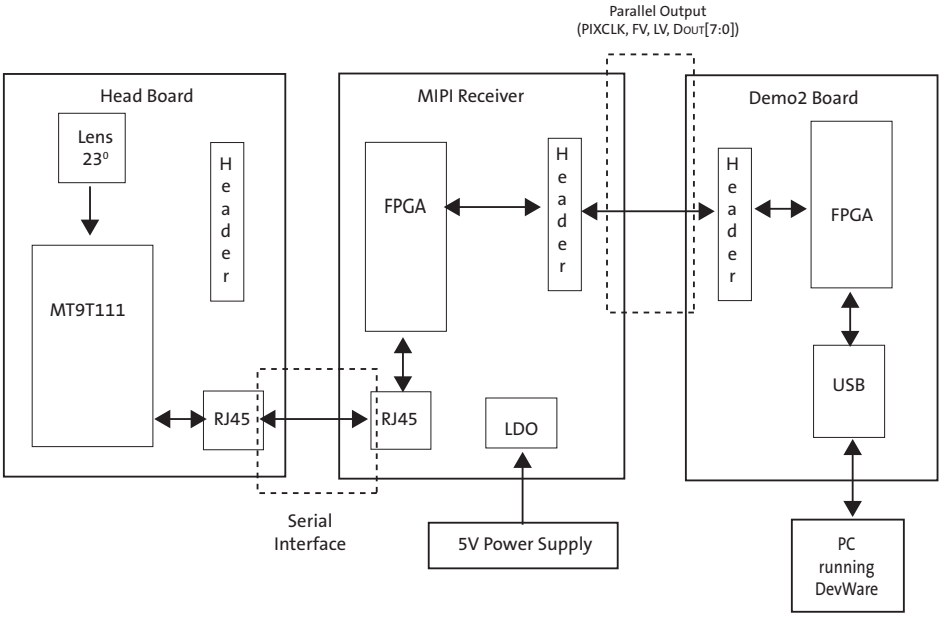


Figure 70: Demo Board (Serial Mode)



Revision History

Rev. C 9/10

- Updated to non-confidential

Rev. B 6/10

- Updated to Aptina template

Rev. A, Preliminary 12/07

- Released as Preliminary to keep in step with data sheet
- Initial release

10 Eunos Road 8 13-40, Singapore Post Center, Singapore 408600 prodmktg@aptina.com www.aptina.com