1/3.2-Inch System-On-A-Chip (SOC) CMOS Digital Image Sensor

MT9D111

Features

- DigitalClarity™ CMOS Imaging Technology
- Superior low-light performance
- Ultra-low-power, low-cost
- Internal master clock generated by on-chip phase-locked loop oscillator (PLL)
- Electronic rolling shutter (ERS), progressive scan
- Integrated image flow processor (IFP) for single-die camera module
- Automatic image correction and enhancement, including lens shading correction
- Arbitrary image decimation with anti-aliasing
- Integrated real-time JPEG encoder
- Integrated microcontroller for flexibility
- Two-wire serial interface providing access to registers and microcontroller memory
- Selectable output data format: ITU-R BT.601 (YCbCr), 565RGB, 555RGB, 444RGB, JPEG 4:2:2, JPEG 4:2:0, and raw 10-bit
- Output FIFO for data rate equalization
- Programmable I/O slew rate
- Xenon and LED flash support with fast exposure adaptation
- Flexible support for external auto focus, optical zoom, and mechanical shutter

Applications

- Cellular phones
- PC cameras
- PDAs

Table 1: Key Performance Parameters

Parameter		Value
Optical format		1/3.2-inch (4:3)
Full resolution		1,600 x 1,200 pixels (UXGA)
Pixel size		2.8µm x 2.8µm
Active pixel array area		4.73mm x 3.52mm
Shutter type		Electronic rolling shutter (ERS) with global reset
Maximum frame rate		15 fps at full resolution, 30 fps in preview mode, (800 x 600)
Maximum data rate/ master clock		80 MB/s 6 MHz to 80 MHz
Supply voltage	Analog	2.5V–3.1V
	Digital	1.7V–1.95V
	I/O	1.7V–3.1V
	PLL	2.5V–3.1V
ADC resolution		10-bit, on-die
Responsivity		1.0/lux-sec (550nm)
Dynamic range		71dB
SNRMAX		42.3dB
Power consumption		348mW at 15 fps, full resolution
		223mW at 30 fps, preview mode
Operating temperature		-30°C to +70°C
Package		Bare die, 64-ball iCSP

Ordering Information

Table 2: Available Part Numbers

Part Number	Description
MT9D111	64-ball iCSP
MT9D111D00STCK15LC1	Bare die
MT9D111W00STCK15LC1	Wafer

Table of Contents

List of Figures

List of Tables

General Description

Micron® Imaging MT9D111 is a 1/3.2 inch 2-megapixel CMOS image sensor with an integrated advanced camera system. The camera system features a microcontroller (MCU) and a sophisticated image flow processor (IFP) with a real-time JPEG encoder. It also includes a programmable general purpose I/O module (GPIO), which can be used to control external auto focus, optical zoom, or mechanical shutter.

The microcontroller manages all components of the camera system and sets key operation parameters for the sensor core to optimize the quality of raw image data entering the IFP. The sensor core consists of an active pixel array of 1668 x 1248 pixels, programmable timing and control circuitry including a PLL and external flash support, analog signal chain with automatic offset correction and programmable gain, and two 10-bit A/D converters (ADC). The entire system-on-a-chip (SOC) has ultra-low power requirements and superior low-light performance that is particularly suitable for mobile applications.

The excellent low-light performance of MT9D111 is one of the hallmarks of DigitalClarity™—Micron's breakthrough low-noise CMOS imaging technology that achieves CCD image quality (based on signal-to-noise ratio and low-light sensitivity) while maintaining the inherent size, cost, power consumption, and integration advantages of CMOS.

Feature Overview

The MT9D111 is a color image sensor with a Bayer color filter arrangement. Its basic characteristics are described in Table 1, "Key Performance Parameters," on page 1.

The MT9D111 has an embedded phase-locked loop oscillator (PLL) that can be used with the common wireless system clock. When in use, the PLL adjusts the incoming clock frequency, allowing the MT9D111 to run at almost any desired resolution and frame rate. To reduce power consumption, the PLL can be bypassed and powered down.

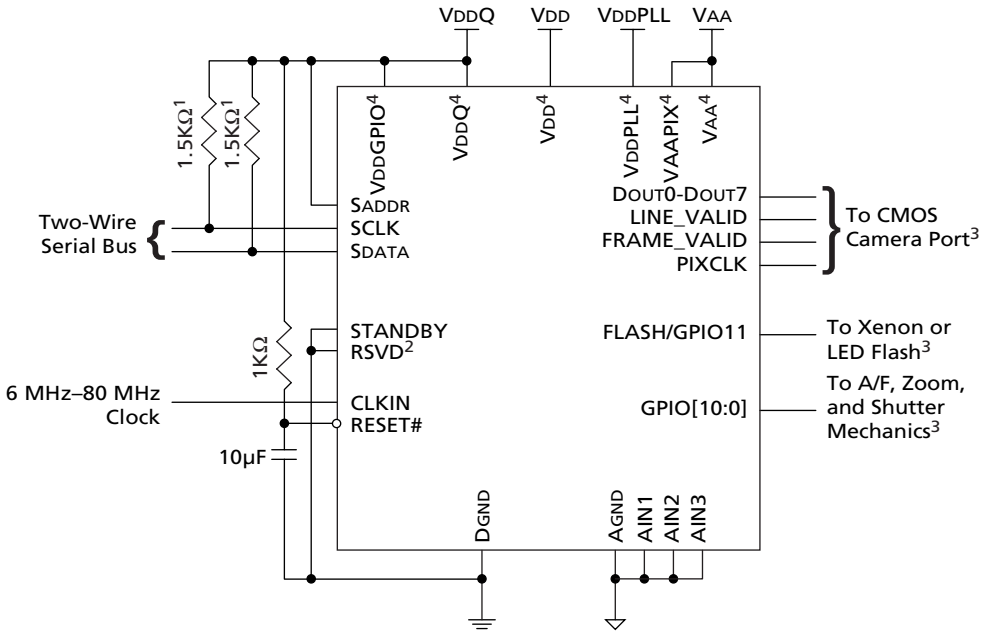
Low power consumption is a very important requirement for all components of wireless devices. The MT9D111 has numerous power conserving features, including an ultra-low power standby mode and the ability to individually shut down unused digital blocks.

Another important consideration for wireless devices is their electromagnetic emission or interference (EMI). The MT9D111 has a programmable I/O slew rate to minimize its EMI and an output FIFO to eliminate output data bursts.

The advanced IFP and flexible programmability of the MT9D111 provide a variety of ways to enhance and optimize the image sensor performance. Built-in optimization algorithms enable the MT9D111 to operate at factory settings as a fully automatic, highly adaptable camera. However, most of its settings are user-programmable.

Typical Connection

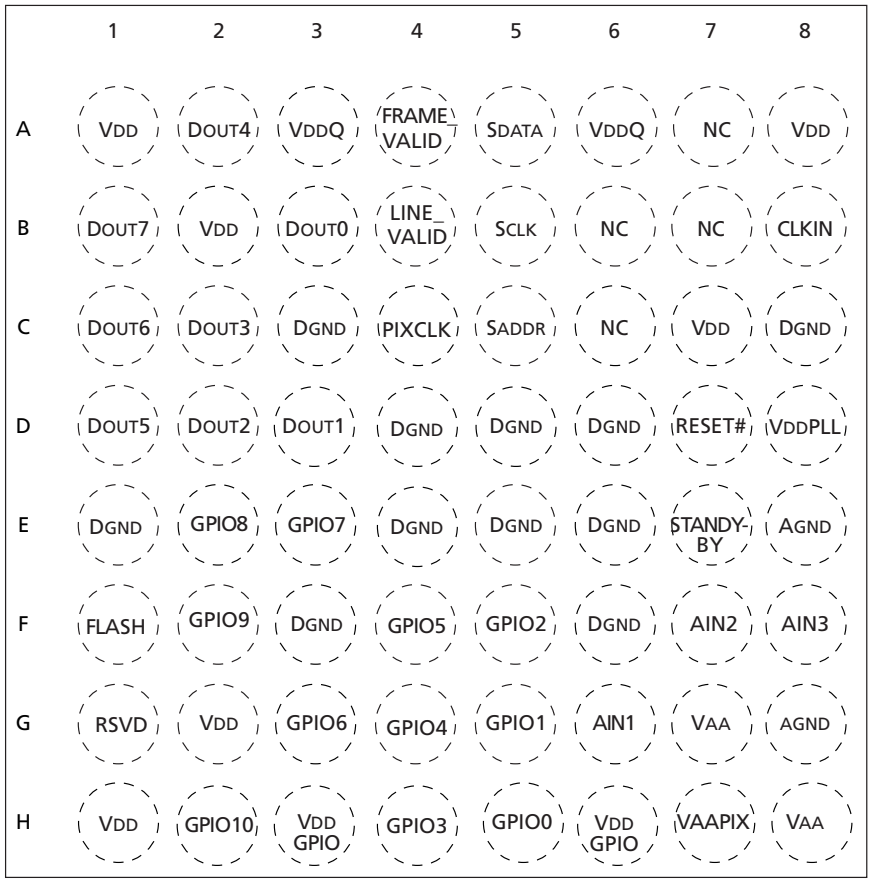
Figure 1: Typical Configuration (Connection)



Note:
1. Resistor value 1.5KΩ is recommended, but may be greater for slower two-wire speed.
2. RSVD must be connected to digital ground for normal device operation.
3. See "Standby Hardware Configuration" on page 153.
4. All power supply pads must be used.

Ballout and Interface

Figure 2: Ball Assignment



Top View
(Ball Down)

Note: All power supply pads must be used.

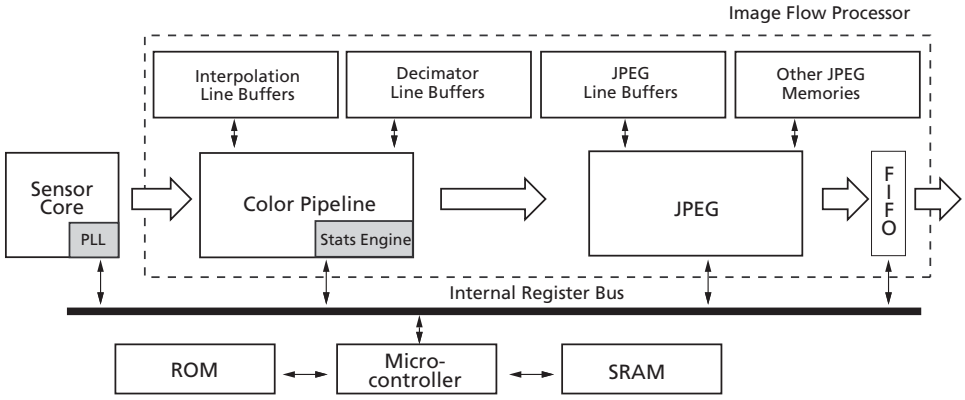
Table 3: Signal Description

Name	Type	Description	Note
CLKIN	Input	Master clock signal (can either drive the on-chip PLL or bypass it).	
RESET#	Input	Master reset signal, active LOW.	
STANDBY	Input	Controls sensor's standby mode.	
RSVD	Input	Reserved for factory test. Tie to digital ground during normal operation.	
AIN1	Input	Analog sampling and test. During normal operation, can be used to feed an external analog signal to an ADC in the sensor core, in order to have the signal sampled during horizontal blanking times and stored in a register.	
AIN2	Input	Analog sampling and test. Can be used like AIN1 during normal operation.	
AIN3	Input	Analog sampling and test. Can be used like AIN1 during normal operation.	
SCLK	Input	Two-wire serial interface clock.	
SADDR	Input	Selects device address for the two-wire serial interface. The address is 0x90 when SADDR is tied LOW, 0xBA if tied HIGH. See also R0x0D:0[10].	
DOUT0-DOUT7	Input	Eight-bit image data output or most significant bits (MSB) of 10-bit sensor bypass mode.	1
FRAME_VALID	Input	Identifies rows in the active image.	1
LINE_VALID	Input	Identifies lines in the active image.	1
PIXCLK	Input	Pixel clock. To be used for sampling DOUT, FRAME_VALID, and LINE_VALID.	1
SDATA	I/O	Two-wire serial interface data.	
GPIO[7:0]	I/O	General purpose digital I/O. Each bit can be independently configured as an input or output. Outputs are controlled by register-programmable waveform generator or by writing to registers GPIO_DATA_L and GPIO_DATA_H. Inputs can be sensed by reading the same registers.	1
FLASH/GPIO11	I/O	GPIO11 or signal to control Xenon or LED flash.	1
GPIO10/STROBE	I/O	GPIO10 or signal to control mechanical shutter.	1
GPIO9/DOUT_LSB	I/O	GPIO9 during normal IFP operation or data bit 1 in 10-bit sensor bypass mode.	1
GPIO8DOUT_LSB0	I/O	GPIO8 during normal IFP operation or data bit 0 in 10-bit sensor bypass mode.	1
VDD	Supply	Digital power (1.8V).	
VDDPLL	Supply	PLL power (2.8V).	
VAA	Supply	Analog power (2.8V).	
VAAPIX	Supply	Pixel array power (2.8V).	
VDDQ	Supply	I/O power (nominal 1.8V or 2.8V).	
VDDGPIO	Supply	I/O power for GPIO (nominal 1.8V or 2.8V).	
AGND	Supply	Analog ground.	
DGND	Supply	Digital, I/O, and PLL ground.	
NC	—	No connect.	

Note: 1. See "Standby Hardware Configuration" on page 153.

Architecture Overview

Figure 3: Block Diagram

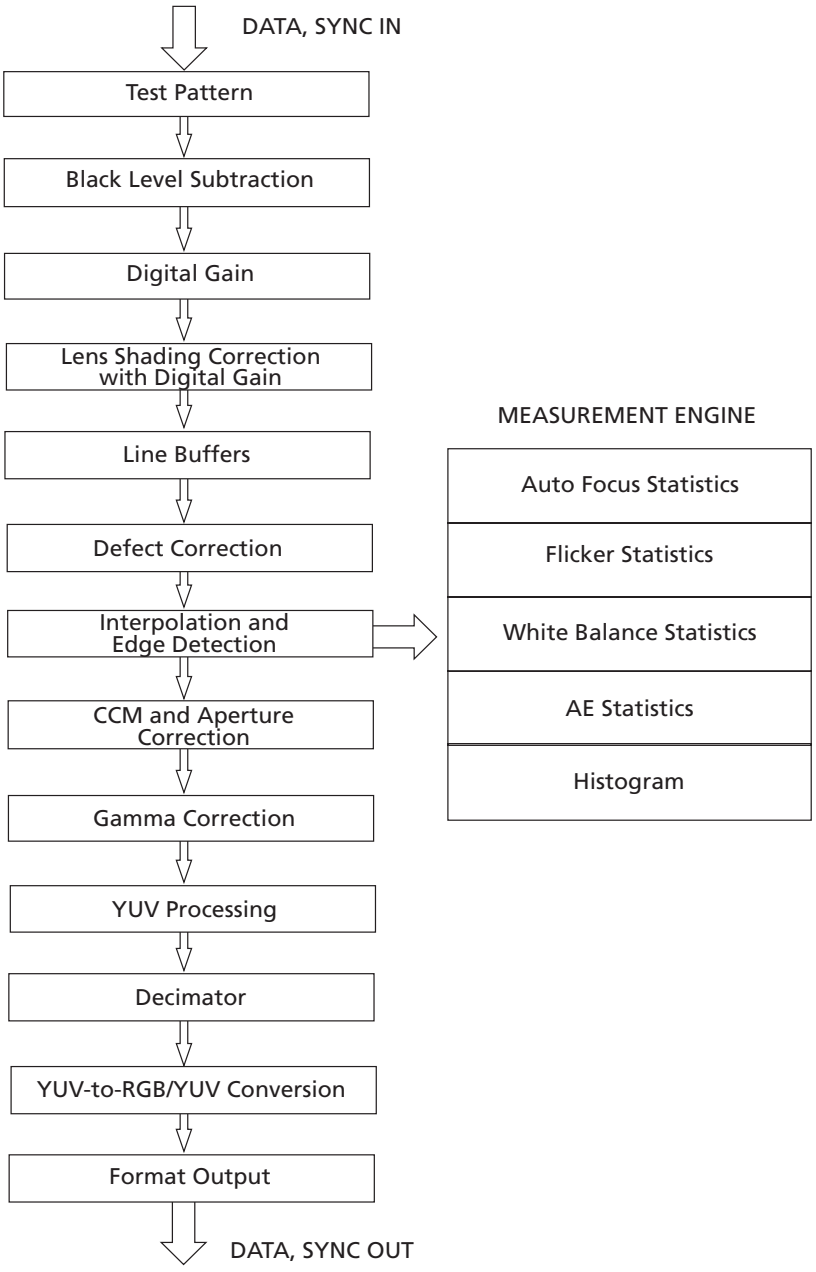


Sensor Core

The MT9D111 sensor core is based on Micron's MT9D011, a stand-alone, 2-megapixel CMOS image sensor with a 2.8µm pixel size. Both image sensors have the same optical size (1/3.2 inches) and maximum resolution (UXGA). Like the MT9D011, the MT9D111 sensor core includes a phase-locked loop oscillator (PLL), to facilitate camera integration and minimize the system cost for wireless and mobile applications. When in use, the PLL generates internal master clock signal whose frequency can be set higher than the frequency of external clock signal CLKIN. This allows the MT9D111 to run at any desired resolution and frame rate up to the specified maximum values, irrespective of the CLKIN frequency.

Color Pipeline

Figure 4: Color Pipeline



Test Pattern

During normal operation of MT9D111, a stream of raw image data from the sensor core is continuously fed into the color pipeline. For test purposes, this stream can be replaced with a fixed image generated by a special test module in the pipeline. The module provides a selection of test patterns sufficient for basic testing of the pipeline.

Black Level Conditioning and Digital Gain

Image stream processing starts with black level conditioning and multiplication of all pixel values by a programmable digital gain.

Lens Shading Correction

Inexpensive lenses tend to produce images whose brightness is significantly attenuated near the edges. Chromatic aberration in such lenses can cause color variation across the field of view. There are also other factors causing fixed-pattern signal gradients in images captured by image sensors. The cumulative result of all these factors is known as lens shading. The MT9D111 has an embedded lens shading correction (LC) module that can be programmed to precisely counter the shading effect of a lens on each RGB color signal. The LC module multiplies RGB signals by a 2-dimensional correction function F(x,y), whose profile in both x and y direction is a piecewise quadratic polynomial with coefficients independently programmable for each direction and color.

Line Buffers

Several data processing steps following the lens shading correction require access to pixel values from up to 8 consecutive image lines. For these lines to be simultaneously available for processing, they must be buffered. The IFP includes a number of SRAM line buffers that are used to perform defect correction, color interpolation, image decimation, and JPEG encoding.

Defect Correction

The IFP performs on-the-fly defect correction that can mask pixel array defects such as high-dark-current (“hot”) pixels and pixels that are darker or brighter than their neighbors due to photoresponse non uniformity. The defect correction algorithm uses several pixel features to distinguish between normal and defective pixels. After identifying the latter, it replaces their actual values with values inferred from the values of nearest same-color neighbors.

Color Interpolation and Edge Detection

In the raw data stream fed by the sensor core to the IFP, each pixel is represented by a 10-bit integer number, which, to make things simple, can be considered proportional to the pixel’s response to a one-color light stimulus, red, green or blue, depending on the pixel’s position under the color filter array. Initial data processing steps, up to and including the defect correction, preserve the 1-color-per-pixel nature of the data stream, but after the defect correction it must be converted to a 3-colors-per-pixel stream appropriate for standard color processing. The conversion is done by an edge-sensitive color interpolation module. The module pads the incomplete color information available for each pixel with information extracted from an appropriate set of neighboring pixels. The algorithm used to select this set and extract the information seeks the best compromise between maintaining the sharpness of the image and filtering out high-frequency noise. The simplest interpolation algorithm is to sort the nearest 8 neighbors of every pixel into 3 sets, red, green, and blue, discard the set of pixels of the same color as the center pixel (if there are any), calculate average pixel values for the remaining 2 sets, and use the averages in lieu of the missing color data for the center pixel. Such averaging reduces high-frequency noise, but it also blurs and distorts sharp transitions (edges) in the image. To avoid this problem, the interpolation module performs edge detection in the neighborhood of every processed pixel and, depending on its results, extracts color information from neighboring pixels in a number of different ways. In effect, it does low-pass filtering in flat-field image areas and avoids doing it near edges.

Color Correction and Aperture Correction

In order to achieve good color fidelity of IFP output, interpolated RGB values of all pixels are subjected to color correction. The IFP multiplies each vector of three pixel colors by a 3 x 3 color correction matrix. The three components of the resulting color vector are all sums of three 10-bit numbers. Since such sums can have up to 12 significant bits, the bit width of the image data stream is widened to 12 bits per color (36 bits per pixel). The color correction matrix can be either programmed by the user or automatically selected by the auto white balance (AWB) algorithm implemented in the IFP. Color correction should ideally produce output colors that are independent of the spectral sensitivity and color cross-talk characteristics of the image sensor. The optimal values of color correction matrix elements depend on those sensor characteristics and on the spectrum of light incident on the sensor.

To increase image sharpness, a programmable aperture correction is applied to color corrected image data, equally to each of the 12-bit R, G, and B color channels.

Gamma Correction

Like the aperture correction, gamma correction is applied equally to each of the 12-bit R, G, and B color channels. Gamma correction curve is implemented as a piecewise linear function with 19 knee points, taking 12-bit arguments and mapping them to 8-bit output. The abscissas of the knee points are fixed at 0, 64, 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2304, 2560, 2816, 3072, 3328, 3584, 3840, and 4095. The 8-bit ordinates are programmable via IFP registers or public variables of mode driver (ID = 7). The driver variables include two arrays of knee point ordinates defining two separate gamma curves for sensor operation contexts A and B.

YUV Processing

After the gamma correction, the image data stream undergoes RGB to YUV conversion and optionally further corrective processing. The first step in this processing is removal of highlight coloration, also referred to as “color kill.” It affects only pixels whose brightness exceeds a certain pre-programmed threshold. The U and V values of those pixels are attenuated proportionally to the difference between their brightness and the threshold. The second optional processing step is noise suppression by 1-dimensional low-pass filtering of Y and/or UV signals. A 3- or 5-tap filter can be selected for each signal.

Image Cropping and Decimation

To ensure that the size of images output by MT9D111 can be tailored to the needs of all users, the IFP includes a decimator module. When enabled, this module performs “decimation” of incoming images, i.e. shrinks them to arbitrarily selected width and height without reducing the field of view and without discarding any pixel values. The latter point merits underscoring, because the terms “decimator” and “image decimation” suggest image size reduction by deleting columns and/or rows at regular intervals. Despite the terminology, no such deletions take place in the decimator module. Instead, it performs “pixel binning”, i.e. divides each input image into rectangular bins corresponding to individual pixels of the desired output image, averages pixel values in these bins and assembles the output image from the bin averages. Pixels lying on bin boundaries contribute to more than one bin average: their values are added to bin-wide sums of pixel values with fractional weights. The entire procedure preserves all image information that can be included in the downsized output image and filters out high-frequency features that could cause aliasing.

The image decimation in the IFP can be preceded by image cropping and/or image decimation in the sensor core. Image cropping takes place when the sensor core is programmed to output pixel values from a rectangular portion of its pixel array - a window -

smaller than the default 1600 x 1200 window. Pixels outside the selected cropping window are not read out, which results in narrower field of view than at the default sensor settings. Irrespective of the size and position of the cropping window, the MT9D111 sensor core can also decimate outgoing images by skipping columns and/or rows of the pixel array, and/or by binning 2 x 2 groups of pixels of the same color. Since decimation by skipping (i.e. deletion) can cause aliasing (even if pixel binning is simultaneously enabled), it is generally better to change image size only by cropping and pixel binning.

The image cropping and decimator module can be used to do digital zoom and pan. If the decimator is programmed to output images smaller than images coming from the sensor core, zoom effect can be produced by cropping the latter from their maximum size down to the size of the output images. The ratio of these two sizes determines the maximum attainable zoom factor. For example, a 1600 x 1200 image rendered on a 160 x 120 display can be zoomed up to 10 times, since 1600/160 = 1200/120 = 10. Panning effect can be achieved by fixing the size of the cropping window and moving it around the pixel array.

YUV-to-RGB/YUV Conversion and Output Formatting

The YUV data stream emerging from the decimator module can either exit the color pipeline as-is or be converted before exit to an alternative YUV or RGB data format. See "Color Conversion Formulas" on page 115 and the description of register R151:1 for more details.

JPEG Encoder and FIFO

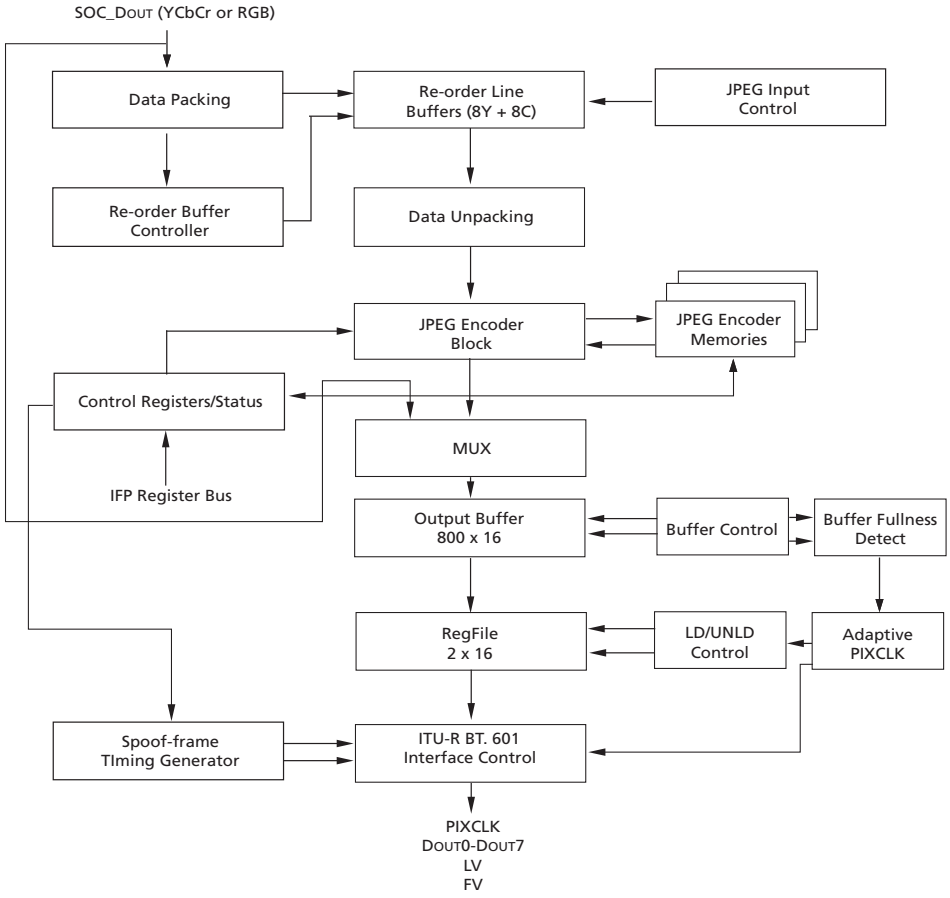
The JPEG compression engine in the MT9D111 is a highly integrated, high-performance solution that can provide sustained data rates of almost 80MB/s for image sizes up to 1600 x 1200. Additionally, the solution provides for low power consumption and full programmability of JPEG compression parameters 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/restart. A flexible configuration and control interface allows for full programmability of various JPEG-specific parameters and tables.

JPEG Encoding Highlights

1. Sequential DCT (baseline) ISO/IEC 10918-1 JPEG-compliant
2. YCbCr 4:2:2 format compression
3. Programmable quantization tables
   - One each for luminance and chrominance (active)
   - Support for three pairs of quantization tables—two pairs serve as a backup for buffer overflow
4. Programmable Huffman Tables
   - 2 AC, 2 DC tables—separate for luminance and chrominance
5. Quality/compression ratio control capability
6. 15 fps MJPEG capability (header processing in external host processor)

Figure 5: JPEG Encoder Block Diagram



Output Buffer Overflow Prevention

The MT9D111 integrates SRAM for the storage of JPEG data. In order to prevent output buffer overflow, the MT9D111 implements an adaptive pixel clock (PIXCLK) rate scheme. When the adaptive pixel clock rate scheme is enabled, PIXCLK can run at clock frequencies of (CLKIN freq/n1), (CLKIN freq/n2), (CLKIN freq/n3), where n1, n2, n3 are register values programmed by the host via the two-wire serial interface. A clock divider block from the master clock CLKIN generates the three clocks, PCLK1, PCLK2, and PCLK3.

At the start of the frame encode, PIXCLK is sourced by PCLK1. The buffer fullness detection block of the SOC switches PIXCLK to PCLK2 and then to PCLK3, if necessary, based on the watermark at the output buffer (i.e., percentage filled up). When the output buffer watermark reaches 50 percent, PIXCLK switches to PCLK2. This increase in PIXCLK rate unloads the output buffer at a higher rate. However, depending on the image complexity and quantization table setting, the compressed image data may still be generated by the JPEG encoder faster than PIXCLK can unload it. Should the output buffer watermark equal 75 percent or higher, PIXCLK is switched to PCLK3. When the output buffer watermark drops back to 50 percent, PIXCLK is switched back to PCLK2. When the output buffer watermark drops to 25 percent, PIXCLK is switched to PCLK1.

When a decision to adapt PIXCLK frequency is made, LINE_VALID, which qualifies the 8-bit data output (DOUT), is de-asserted until PIXCLK is safely switched to the new clock. LINE_VALID is independent of the horizontal timing of the uncompressed imaged. Its assertion is strictly based on compressed image data availability.

Should an output buffer overflow still occur with PIXCLK at the maximum frequency, the output buffer and the small asynchronous FIFO is flushed immediately. This causes LINE_VALID to be de-asserted. FRAME_VALID is also de-asserted.

In addition to the adaptive PIXCLK rate scheme, the MT9D111 also has storage for 3 sets of quantization tables (6 tables). In the event of output buffer overflow during the compression of the current frame, another set of the preloaded quantization tables can be used for the encoding of the immediate next frame. Then, the MT9D111 starts compressing the next frame starting with the nominal PIXCLK frequency.

Control (Two-Wire Serial Interface)

Camera control and JPEG configuration/control are accomplished via a two-wire serial interface. The interface supports individual access to all camera function registers and JPEG control registers. In particular, all tables located in the JPEG quantization and Huffman memories are accessible via the two-wire interface. To write to a particular register, the external host processor must send the MT9D111 device address (selected by SADDR or R0x0D:0[10]), the address of the register, and data to be written to it. See "Appendix A: Two-Wire Serial Register Interface" on page 180 for a description of read sequence and for details of the two-wire serial interface protocol.

Data

JPEG data is output in a BT656-like 8-bit parallel bus DOUT0-DOUT7, with FRAME_VALID, LINE_VALID, and PIXCLK. JPEG output data is valid when both FRAME_VALID and LINE_VALID are asserted. When the JPEG data output for the frame completes, or buffer overflow occurs, LINE_VALID and FRAME_VALID are de-asserted. The output clock runs at frequencies selected by frequency divisors N1, N2, and N3 (registers R0x0E:2 and R0x0F:2), depending on output buffer fullness.

Auto Focus

Algorithm

Auto focus (AF) algorithm implemented in the MT9D111 firmware seeks to maximize sharpness of vertical lines in images output by the sensor, by guiding an external lens actuator to the position of best lens focus. The algorithm is actuator-independent: it provides guidance by means of an abstract 1-dimensional position variable, leaving the translation of its changes into physical lens movements to a separate AF mechanics (AFM) driver. The AF algorithm relies on the AFM driver to generate digital output signals needed to move different lens actuators and to correctly indicate at all times if the lens is stationary or moving. The latter is required to prevent the AF algorithm from using line sharpness measurements distorted by concurrent lens motion.

For measuring line sharpness, the AF algorithm relies on focus measurement engine in the color pipeline, which is a programmable vertical-edge-filtering module. The module convolves two pre-programmed 1-dimensional digital filters with luminance (Y) data it receives row by row from the color interpolation module. In every interpolated image, the pixels whose Y values are used in the convolution form a rectangular block that can be arbitrarily positioned and sized, and in addition divided into up to 16 equal-size sub-blocks, referred to as AF windows or zones. The absolute values of convolution results are summed separately for each filter over each of the AF windows, yielding up to 32 sums per frame. As soon as these sums or raw sharpness scores are computed, they are put in dedicated IFP registers (R[77:84]:2 and R[87:94]:2), as are Y averages from all the AF windows (in R[67:74]:2). The AF algorithm reduces these data to one normalized sharpness score per AF window, by calculating for each window the ratio (S1+S2)/<Y>, where <Y> is the average Y and S1 and S2 are the raw sharpness scores from the two filters multiplied by 128. Programming the filters into the MT9D111 includes specifying their relative weights, so each ratio can be called a weighted average of two equally normalized sharpness scores from the same AF window. In addition to unequal weighting of the filters, the AF algorithm permits unequal weighting of the windows, but window weights are not included in the normalized sharpness scores, for a reason that will soon become clear.

There are several motion sequences through which the MT9D111 AF algorithm can bring a lens to best focus position. All these sequences begin with a jump to a preselected start position, e.g. the infinity focus position. This jump is referred to as the first flyback. It is followed by a unidirectional series of steps that puts the lens at up to 19 preselected positions different from the start position. This series of steps is called the first scan.

Before and during this scan, the AF algorithm stops the lens at each preselected position for long enough to obtain valid sharpness scores. The first normalized score from each AF window is stored as both the worst (minimum) and best (maximum) score for that window. These two extreme scores are then updated as the lens moves from one position to the next and a new maximum position is memorized at every update of the maximum score. In effect, the preselected set of lens positions is scanned for maxima of the normalized sharpness scores, while at the same time information needed to validate each maximum is being collected. This information is in the difference between the maximum and the minimum of the same score. A small difference in their values indicates that the score is not sensitive to the lens position and therefore its observed extrema are likely determined by random noise. On the other hand, if the score varies a lot with the lens position, its maximum is much more likely to be valid, i.e. close to the true sharpness maximum for the corresponding AF window. Due to these considerations, the AF algorithm implemented in MT9D111 ignores the maxima of all sharpness scores whose peak-to-trough variation is below a preset percentage threshold. The remaining max-

ima, if any, are sorted by position and used to build a weight histogram of the scanned positions. The histogram is build by assigning to each position the sum of weights of all AF windows whose normalized sharpness scores peaked at that position. The position with the highest weight in the histogram is then selected as the best lens position. This method of selecting the best position may be compared to voting. The voting entities are the AF windows, i.e. different image zones. Depending on the imaged scene, they may all look sharp at the same lens position or at different ones. If all the zones have equal weight, the lens position at which a simple majority of them looks sharp is voted the best. If the weights of the zones are unequal, it means that making some zones look sharp is more important than maximizing the entire sharp-looking area in the image. If there are no valid votes, because sharpness scores from all the AF windows vary too little with the lens position, the AF algorithm arbitrarily chooses the start position as the best.

What happens after the first scan is user-programmable—the AF algorithm gives the user a number of ways to proceed with final lens positioning. The user should select a way that best fits the magnitude of lens actuator hysteresis and desired lens proximity to the truly optimal position. Actuators with large, unknown or variable hysteresis should do a second flyback and either jump or retrace the steps of the first scan to the best scanned position. Actuators with constant hysteresis (like gear backlash) can be moved to that position directly from the end position of the scan—the AF algorithm offers an option to automatically increase the length of this move by a preprogrammed backlash-compensating step. Finally, if the first scan is coarse relative to the positioning precision of the lens actuator and depth of field of the lens, an optional second fine scan can be performed around the lens position voted best after the first scan. This second scan is done in the same way as the first, except that the positions it covers are not pre-selected. Instead, the AF algorithm user must set step size and number of steps for the second scan. The second scan must be followed by the same hysteresis-matching motion sequence as the first scan, e.g. a third flyback and jump to the best position.

Modes

There are four AF camera modes that the MT9D111 can fully support if it controls the position of the camera lens.

1. Snapshot mode
   In this mode, a camera performs auto focusing upon a user command to do so. When the auto focusing is finished, a snapshot is normally taken and there is no further AF activity until next appropriate user command. The MT9D111 can do the auto focusing using its own AF algorithm described above or a substitute algorithm loaded into its RAM. It can then wait or automatically proceed with other operations required to take a snapshot.
2. Locked mode
   The MT9D111 can be commanded to lock the lens in its current position. Between the command to lock the lens and another to release it, the lens does not respond to other commands or scene changes.
3. Focus-free mode
   In many situations, e.g. under low light or during video recording, it may be impossible or undesirable to focus the lens prior to every image capture. Instead, the lens can be locked in a position most likely to produce satisfactory images, e.g. the hyperfocal position. This position can be programmed into the MT9D111, and it can move and hold the lens there on command.
4. Manual mode
   In this mode there is no AF activity—focusing the camera is left to its user. The user typically can move the camera lens in steps, by manually issuing commands to the lens actuator, and observe the effect of his actions on a preview display. The MT9D111

can provide 30 fps image input for the display and simultaneously translate user commands received via two-wire serial interface into digital waveforms driving the lens actuator.

Lens Actuator Interface

Actuators used to move lenses in AF cameras can be classified into several broad categories that differ significantly in their requirements for driving signals. These requirements also vary from one device to another within each category. To ensure its compatibility with many different actuators, the MT9D111 includes a general purpose input/output module (GPIO).

In essence, the GPIO is a programmable rectangular waveform generator, with 12 individually controllable output pads (GPIO0 through GPIO11), a separate power supply pad (V_{DD}GPIO), and a separate clock domain that can be disconnected from the master clock to save power when the GPIO is not in use. The GPIO can toggle its output pads as fast as half the master clock frequency (every 25ns at 80 MHz).

An external host processor and the embedded microcontroller (MCU) of the MT9D111 have two ways to control the voltages on the GPIO output pads:

1. Setting or clearing bits in a control register
   The state of the GPIO pads is updated immediately after writing to the register is finished. Since writing via the two-wire serial interface takes some time, this way does not give the host processor a very precise control over GPIO output timing.
2. Waveform programming
   The second way to obtain a desired output from the GPIO is to program into its registers a set of periodic waveforms and initialize their generation. The GPIO then generates the programmed waveforms on its own, without waiting for any further input, and therefore with the best attainable timing precision. If necessary, the GPIO can notify the MCU and the host processor about reaching certain points in the waveforms generation, e.g., the end of a particular waveform. Every GPIO notification has two components: the GPIO sends a wakeup signal to the MCU and sets a bit in its status register that can be polled by the MCU and/or the host processor. The wakeup signals have an effect only when the MCU is in sleep mode.

The MT9D111 can be set up not only to output digital signals to a lens actuator and/or other similar devices, but also to receive their digital and analog feedback. All GPIO output pads are reconfigurable as high-impedance digital inputs. The logical state of each GPIO pad is mirrored by the state of a bit in a dedicated register, which allows the MCU and host processor to sample digital input signals at intervals equal to their respective register read times. Analog input signals (0.1V to 1.0V) can be sampled using one of the 10-bit ADCs in the sensor core. During horizontal blanking periods, when it does not digitize the sensor signal, this ADC samples voltages on AIN1, AIN2, and AIN3 input pads. The results are stored in dedicated registers. The maximum signal sampling rate permitted by this scheme is about 36000 samples per second.

Context and Operational Modes

The MT9D111 can operate in several modes, including preview, still capture (snapshot), and video. All modes of operation are individually configurable and are organized as two contexts—context A and context B. A context is defined by sensor image size, frame rate, resolution and other associated parameters. The user can switch between the two contexts by sending a command via the two-wire serial interface.

Preview

Context A is primarily intended for use in the preview mode. During preview, the sensor usually outputs low resolution images at a relatively high frame rate, and its power consumption is kept to a minimum. Context B can be configured for the still capture or video mode, as required by the user. For still capture configuration, the user typically specifies the desired output image size, if JPEG compression and flash should be enabled, how many frames to capture, etc. For video, the user might select a different image size and a fixed frame rate.

Snapshot and Flash

To take a snapshot, the user must send a command that changes the context from A to context B. Typical sequence of events after this command is as follows. First, the camera may turn on its LED flash, if it has one and is required to use it. With the flash on, the camera exposure and white balance is automatically adjusted to the changed illumination of the scene. Next, the camera performs auto focusing. Once in focus, it enables JPEG compression and capture one or more frames of desired size. A camera equipped with a Xenon flash strobes it during the capture. Completing the sequence, the camera automatically returns to context A and resume running preview.

Video

To start video capture, the user has to change relevant context B settings, such as capture mode, image size and frame rate, and again send a context change command. Upon receiving it, the MT9D111 switches to the modified context B settings, while continuing to output YUV-encoded image data. Auto exposure and auto focus automatically switches to smooth continuous operation. To exit the video capture mode, the user has to send another context change command causing the sensor to switch back to context A.

Auto Exposure

The auto exposure (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.

Two auto exposure algorithm modes are available:

1. preview
2. scene evaluative

Auto exposure is implemented by means of a firmware driver that analyzes image statistics collected by exposure measurement engine, 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.

Preview Mode

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

The backlight compensation is achieved by weighting the luminance in the center of the image higher than the luminance on the periphery. Other algorithm features include the rejection of fast fluctuations in illumination (time averaging), control of 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.

Scene Evaluative Algorithm

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

Auto White Balance

The MT9D111 has a built-in auto white balance (AWB) algorithm designed to compensate for the effects of changing spectra of the scene illumination on the quality of the color rendition. This sophisticated algorithm consists of two major parts: a measurement engine performing statistical analysis of the image and a driver performing the selection of the optimal color correction matrix, digital, and sensor core analog gains. While default settings of these algorithms are adequate in most situations, the user can re-program base color correction matrices, place limits on color channel gains, and control the speed of both matrix and gain adjustments. Unlike simple white balancing algorithms found in many PC cameras, the MT9D111 AWB does not require the presence of gray or white elements in the image for good color rendition. The AWB does not attempt to locate "brightest" or "grayest" element of the image but instead performs sophisticated image analysis to differentiate between changes in predominant spectra of illumination and changes in predominant colors of the scene. While defaults are suitable for most applications, a wide range of algorithm parameters can be overwritten by the user via the serial interface.

Flicker Detection

Flicker occurs when the integration time is not an integer multiple of the period of the light intensity. The automatic flicker detection block does not compensate for the flicker, but rather avoids it by detecting the flicker frequency and adjusting the integration time. For integration times below the light intensity period (10ms for 50Hz environment), flicker cannot be avoided.

Registers and Variables

Four types of configuration controls are available:

1. Hardware registers
2. Driver variables
3. Special function registers (SFR)
4. MCU SRAM

The following convention is used in the text below to designate registers and variables:

R0x12:1, R0x12:1[3:0] or R18:1, R18:1[3:0]

These refer to two-wire accessible register number 18, or 0x12 hexadecimal, located on page 1. [3:0] indicate bits. Registers numbers range 0..255 and bits range 15..0.

- ae.Target
  This refers to variable ‘Target” in the AE driver.
- SFR 0x1080 or SRAM 0x0400
  This refers to special function register or SRAM located at address 0x1080 in MCU memory space.

How to Access

Registers, variables, and SFRs are accessed in different ways.

Registers

Hardware registers are organized into several pages. Page 0 contains sensor controls. Page 1 contains color pipeline controls. Page 2 contains JPEG, output FIFO and more color pipeline controls. The desired page is selected by writing the desired value to R0xF0. After that all READs and WRITEs to registers 0..255 except R0xF0 and R0xF1, is directed to the selected page. R0xF0 and R0xF1 are special registers and are present on all pages. See “Appendix A: Two-Wire Serial Register Interface” on page 180 for description of two-wire register access.

Variables

Variables are located in the microcontroller RAM memory. Each driver, such as auto exposure, white balance, auto focus, etc., has a unique driver ID (0..31) and a set of public variables organized as a structure. Each variable in this structure is uniquely identified by its offset from the top of the structure and its size. The size can be 1 or 2 bytes, while the offset is 1 byte.

All driver variables (public and private) can be accessed via R198:1 and R200:1. While two access modes are available-access by physical address and by logical address, the public variables are typically accessed by the logical method. The logical address, which is set in R198:1, consists of a 5-bit driver ID number and a variable offset. Examples are provided below.

To set the variable ae.Target=50:

- The variable has a driver ID of 2. Therefore, set R198:1[12:8]=2
- The variable has an offset of 6. Therefore, set R198:1[7:0]=6
- This is a logical access. Therefore, set R198:1[14:13]=01
- The size of the variable is 8 bits. Therefore, set R198:1[15]=1
- By combining these bits, R198:1=0xA206.
- Set R200:1=50 for the value of the variable

To read the variable ae.Target:

- Since this is the same variable as the above example, R198:1=0xA206

- Read R200:1 for the current variable value

To set the variable mon.arg1=0x1234:

- The variable has a driver ID of 0. Therefore, set R198:1[12:8]=0
- The variable has an offset of 3. Therefore, set R198:1[7:0]=3
- This is a logical access. Therefore, set R198:1[14:13]=01
- The size of the variable is 16 bits. Therefore, set R198:1[15]=0
- By combining these bits, R198:1=0x2003
- Set R200:1=0x1234 for the value of the variable

To read the variable mon.arg1:

- Since this is the same variable as the above example, R198:1=0x2003
- Read R200:1 for the current variable value

Please see description for R198:1 and R200:1 in Table 7, "IFP Registers, Page 2," on page 52 for more details.

Special Function Registers (SFR) and MCU SRAM

Special function registers (SFR) are registers connected to the local bus of the micro-controller. These registers include GPIO, waveform generator, and those important for firmware operation. SFR are accessed by physical address. MCU SRAM consists of 1K system memory and 1K user memory. Examples of access:

- Write into user SRAM. Use to upload code
  a. R198:1 = 0x400 // address
  b. R200:1 = 0x1234 // write 16-bit value
- Read from user SRAM
  c. R198:1 = 0x400 // address
  d. Read R200:1 // read 16-bit value
- Write to 8-bit GPIO register
  e. R198:1 = 0x9079 // GPIO_DIR_L at 0x1079
  f. R200:1 = 0x00FE // Configure GPIO[0] as output
- Read from 8-bit GPIO register
  g. R198:1 = 0x9079 // GPIO_DIR_L at 0x1079
  h. Read R200:1 //Check GPIO[7:0] pad state

See R198:1 and R200:1 description in Table 7, "IFP Registers, Page 2," on page 52 for more detail.

Registers

Sensor Core Registers

Table 4: Sensor Core Register Defaults

ADDR	Register Description	Default PRE MCU BOOT	Default AFTER MCU BOOT
0x00	Reserved	0x1519	0x1519
0x01	Row Start	0x001C	0x001C
0x02	Column Start	0x003C	0x003C
0x03	Row Width	0x04B0	0x04B0
0x04	Col Width	0x0640	0x0640
0x05	Horizontal Blanking B	0x015C	0x0204
0x06	Vertical Blanking B	0x0020	0x002F
0x07	Horizontal Blanking A	0x00AE	0x00FE
0x08	Vertical Blanking A	0x0010	0x000C
0x09	Shutter Width	0x04D0	N/A
0x0A	Row Speed	0x00011	0x0001
0x0B	Extra Delay	0x0000	0x0000
0x0C	Shutter Delay	0x0000	0x0000
0x0D	Reset	0x0000	0x0000
0x1F	Frame Valid Control	0x0000	0x0000
0x20	Read Mode B	0x0000	0x0300
0x21	Read Mode A	0x0490	0x8400
0x22	Dark Col/Rows	0x010F	0x010F
0x23	Flash	0x0608	0x0608
0x24	Extra Reset	0x8000	0x8000
0x25	Line Valid Control	0x0000	0x0000
0x26	Bottom Dark Rows	0x0007	0x0007
0x2B	Green1 Gain	0x0020	N/A
0x2C	Blue Gain	0x0020	N/A
0x2D	Red Gain	0x0020	N/A
0x2E	Green2 Gain	0x0020	N/A
0x2F	Global Gain	0x0020	N/A
0x30	Row Noise	0x042A	0x042A
0x59	Black Rows	0x00FF	0x00FF
0x5B	Dark G1 average	N/A	N/A
0x5C	Dark B average	N/A	N/A
0x5D	Dark R average	N/A	N/A
0x5E	Dark G2 average	N/A	N/A
0x5F	Calib Threshold	0x231D	0x231D
0x60	Calib Control	0x0080	0x0080
0x61	Calib Green1	0x0000	0x0000
0x62	Calib Blue	0x0000	0x0000
0x63	Calib Red	0x0000	0x0000
0x64	Calib Green2	0x0000	0x0000
0x65	Clock Control	0xE000	0xE000
0x66	PLL Control 1	0x2809	0x1000

Table 4: Sensor Core Register Defaults (continued)

ADDR	Register Description	Default PRE MCU BOOT	Default AFTER MCU BOOT
0x67	PLL Control 2	0x0501	0x0500
0xC0	Global Shutter Control	0x0000	0x0000
0xC1	Start Integration (T1)	0x0064	0x0064
0xC2	Start Readout (T2)	0x0064	0x0064
0xC3	Assert Strobe (T3)	0x0096	0x0096
0xC4	De-assert Strobe (T4)	0x00C8	0x00C8
0xC5	Assert Flash	0x0064	0x0064
0xC6	De-assert Flash	0x0078	0x0078
0xE0	External Sample 1	0x0000	0x0000
0xE1	External Sample 2	0x0000	0x0000
0xE2	External Sample 3	0x0000	0x0000
0xE3	External Sampling Control	0x0000	0x0000
0xF0	Page Register	0x0000	0x0000
0xF1	Bytewise Address	0x0000	0x0000
0xF2	Context Control	0x000B	0x0000
0xFF	Reserved	0x1519	0x1519

Registers

Notation used in the sensor register description table:

Sync'd to frame start

N = No. The register value is updated and used immediately.

Y = Yes. The register value is updated at next frame start as long as the synchronize changes bit is 0. Frame start is defined as when the first dark row is read out. By default, this is 8 rows before FRAME_VALID goes HIGH.

Bad frame

A bad frame is a frame where all rows do not have the same integration time, or offsets to the pixel values changed during the frame.

N = No. Changing the register value does not produce a bad frame.

Y = Yes. Changing the register value might produce a bad frame.

YM = Yes, but the bad frame is masked out unless the "show bad frames" feature is (R0x0D:0[8]) is enabled.

Table 5: Sensor Register Description

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
R0—0x00 - Reserved (R/O)					
Bits 15:0	Reserved	Reserved.	1519		
R1—0x01 - Row Start (R/W)					
Bits 10:0	Row Start	The first row to be read out, excluding any dark rows that may be read. To window the image down, set this register to the starting Y value. Setting a value less than 20 is not recommended because the dark rows should be read using R0x22:0.	1C	Y	YM
R2—0x02 - Column Start (R/W)					
Bits 10:0	Column Start	The first column to be read out, excluding dark columns that may be read. To window the image down, set this register to the starting X value. Setting a value below 52 is not recommended because readout of dark columns should be controlled by R0x22:0.	3C	Y	YM
R3—0x03 - Row Width (R/W)					
Bits 10:0	Row Width	Number of rows in the image to be read out, excluding any dark rows or border rows that may be read. The minimum supported value is 2.	4B0	Y	YM
R4—0x04 - Column Width (R/W)					
Bits 10:0	Column Width	Number of columns in image to be read out, excluding any dark columns or border columns that may be read. The minimum supported value is 9 in 1 ADC mode and 17 in 2 ADC mode.	640	Y	YM
R5—0x05 - Horizontal Blanking—Context B (R/W)					
Bits 13:0	Horizontal Blanking—Context B	Number of blank columns in a row when context B is selected (R0xF2:0[0] = 1). The extra columns are added at the beginning of a row. See "Frame Rate Control" on page 128 for more information on supported register values.	15C	Y	YM

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
R6—0x06 - Vertical Blanking—Context B (R/W)					
Bits 14:0	Vertical Blanking—Context B	Number of blank rows in a frame when context B is selected (R0xF2:0[1] = 1). The minimum supported value is (4 + R0x22:0[2:0]). The actual vertical blanking time may be controlled by the shutter width (R0x09:0). See "Raw Data Timing" on page 119.	20	Y	N
R7—0x07 - Horizontal Blanking—Context A (R/W)					
Bits 13:0	Horizontal Blanking—Context A	Number of blank columns in a row when context A is selected (R0xF2:0[0] = 0). The extra columns are added at the beginning of a row. See "Frame Rate Control" on page 128 for more information on supported register values.	AE	Y	YM
R8—0x08 - Vertical Blanking—Context A (R/W)					
Bits 14:0	Vertical Blanking—Context A	Number of blank rows in a frame when context A is chosen (R0xF2:0[1] = 1). The minimum supported value is (4 + R0x22:0[2:0]). The actual vertical blanking time may be controlled by the shutter width (R0x9:0). See "Raw Data Timing" on page 119.	10	Y	N
R9—0x09 - Shutter Width (R/W)					
Bits 15:0	Shutter Width	Integration time in number of rows. The integration time is also influenced by the shutter delay (R0x0C:0) and the overhead time.	4D0	Y	N
R10—0x0A - Row Speed (R/W)					
Bits 15:14	Reserved	Do not change from default value.			
Bit 13	Reserved	Do not change from default value.			
Bit 8	Invert Pixel Clock	Invert PIXCLK. When clear, FRAME_VALID, LINE_VALID, and DOUT are set up relative to the delayed rising edge of PIXCLK. When set, FRAME_VALID, LINE_VALID, and DOUT are set up relative to the delayed falling edge of PIXCLK.	0	N	N
Bits 7:4	Delay Pixel Clock	Number of half master clock cycle increments to delay the rising edge of PIXCLK relative to transitions on FRAME_VALID, LINE_VALID, and DOUT.	1	N	N
Bit 3	Reserved	Do not change from default value..			
Bits 2:0	Pixel Clock Speed	A programmed value of *N* gives a pixel clock period of *N* master clocks in 2 ADC mode and *2*N* master clocks in 1 ADC mode. A value of "0" is treated like (and reads back as) a value of "1."	1	Y	YM
R11—0x0B - Extra Delay (R/W)					
Bits 13:0	Extra Delay	Extra blanking inserted between frames. A programmed value of *N* increases the vertical blanking time by *N* pixel clock periods. Can be used to get a more exact frame rate. It may affect the integration times of parts of the image when the integration time is less than one frame.	0	Y	N^2

Table 5: Sensor Register Description (continued)

Bit Field		Description	Default (Hex)	Sync'd to Frame Start	Bad Frame
R12—0x0C - Shutter Delay (R/W)					
Bits 13:0	Shutter Delay	The amount of time from the end of the sampling sequence to the beginning of the pixel reset sequence. If the value in this register exceeds the row time, the reset of the row does not complete before the associated row is sampled, and the sensor does not generate an image. A programmed value of *N* reduces the integration time by (*N/2*) pixel clock periods in 1 ADC mode and by *N* pixel clock periods in 2 ADC mode.	0	Y	N
R13—0x0D - Reset (R/W)					
Bit 15	Synchronize Changes	By default, update of many registers are synchronized to frame start. Setting this bit inhibits this update; register changes remain pending until this bit is returned to "0." When this bit is returned to "0," all pending register updates are made on the next frame start.	0	N	N
Bit 10	Toggle SADDR	By default, the sensor serial bus responds to addresses 0xBA and 0xBB. When this bit is set, the sensor serial bus responds to addresses 0x90 and 0x91. WRITEs to this bit are ignored when STANDBY is asserted. See "Slave Address" on page 181.	0	N	N
Bit 9	Restart Bad Frames	When set, a restart is forced to take place whenever a bad frame is detected. This can shorten the delay when waiting for a good frame because the delay, when masking out a bad frame, is the integration time rather than the full frame time.	0	N	N
Bit 8	Show Bad Frames	1—Output all frames (including bad frames). 0—Only output good frames (default). A bad frame is defined as the first frame following a change to: window size or position, horizontal blanking, pixel clock speed, zoom, row or column skip, binning, mirroring, or use of border.	0	N	N
Bit 7:6	Inhibit Standby / Drive Pins	00 or 01—setting STANDBY high puts sensor into standby state with high-impedance outputs 10—setting STANDBY high only puts the outputs in High-Z 11—causes STANDBY to be ignored	0	N	N
Bit 5	Reset SOC	When this bit is set to 1, SOC is put in reset state. It exits this state when the bit is set back to 0. Any attempt to access SOC registers (IFP page 1 and 2) in the reset state results in a sensor hang-up. The sensor cannot recover from it without a hard reset or power cycle.	0	N	
Bit 4	Output Disable	Setting this bit to 1 puts the pin interface in a High-Z. See "Output Enable Control" on page 153. If the DOUT*, PIXCLK, Frame_Valid, or Line_Valid is floating during STANDBY, this bit should be set to "0" to turn off the input buffer, reducing standby current (see technical note TN0934 "Standby Sequence"). This bit must work together with bit 6 to take effect.	0		
Bit 3	Reserved	Keep at default value.	0		

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bit 2	Standby	Setting this bit to 1 places the sensor in a low-power state. Any attempt to access registers R[0xF7:0xFD]:0 in this state results in a sensor hang-up. The sensor cannot recover from it without a hard reset or power cycle.	0	N	YM
Bit 1	Restart	Setting this bit causes the sensor to truncate the current frame and start resetting the first row. The delay before the first valid frame is read out is equal to the integration time. This bit is write - 1 but always reads back as 0.	0	N	YM
Bit 0	Reset	Setting this bit puts the sensor in reset; the frame being generated is truncated and the pin interface goes to an idle state. All internal registers (except for this bit) go to the default power-up state. Clearing this bit resumes normal operation.	0	N	YM
R31—0x1F - FRAME_VALID Control (R/W)					
Bit 15	Enable Early FRAME_VALID Fall	1—Enables the early disabling of FRAME_VALID as set in bits 14:8. LINE_VALID is still generated for all active rows. 0—Default. FRAME_VALID goes low 6 pixel clocks after last LINE_VALID.	0	N	N
Bits 14:8	Early FRAME_VALID Fall	When enabled, the FRAME_VALID falling edge occurs within the programmed number of rows before the end of the last LINE_VALID. (1 + bits 14:8)*row time + constant (constant = 3 in default mode) The value of this field must not be larger than row width R0x03:0.	0	N	N
Bit 7	Enable Early FRAME_VALID Rise	1—Enables the early rise of FRAME_VALID as set in bits 6:0. 0—Default. FRAME_VALID goes HIGH 6 pixel clocks before first LINE_VALID.	0	N	N
Bits 6:0	Early FRAME_VALID Rise	When enabled, the FRAME_VALID rising edge is set HIGH the programmed number of rows before the first LINE_VALID: (1 + bits 6:0)*row time + horizontal blank + constant (constant = 3 in default mode).	0	N	N
R32—0x20 - Read Mode—Context B (R/W)					
Bit 15	Binning—Context B	When read mode context B is selected (R0xF2:0[3] = 1): 0—Normal operation. 1—Binning enabled. See "Binning" on page 127 and See "Frame Rate Control" on page 128 for a full description.	0	Y	YM

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bit 13	Zoom Enable	0—Normal operation. 1—Zoom is enabled, with zoom factor [zoom] defined in bits 12:11. In zoom mode, the pixel data rate is slowed by a factor of [zoom]. This is achieved by outputting [zoom - 1] blank rows between each output row. Setting this mode allows the user to fill a window that is [zoom] times larger with interpolated data. The pixel clock speed is not affected by this operation, and the output data for each pixel is valid for [zoom] pixel clocks. Every row is followed by [zoom - 1] blank rows (with their own LINE_VALID, but all data bits = 0) of equal time. The combination of this register and an appropriate change to the window sizing registers allows the user to zoom to a region of interest without affecting the frame rate.	0	Y	YM
Bits 12:11	Zoom	When zoom is enabled by bit 13, this field determines the zoom amount: 00—Zoom 2x 01—Zoom 4x 10—Zoom 8x 11—Zoom 16x	0	Y	YM
Bit 10	Use 1 ADC—Context B	When read mode context B is selected (bit 3, R0xF2:0 = 1): 0—Use both ADCs to achieve maximum speed. 1—Use 1 ADC to reduce power. Maximum readout frequency is now half the master clock frequency, and the pixel clock is automatically adjusted as described for the pixel clock speed register.	0	Y	YM
Bit 9	Show Border	This bit indicates whether to show the border enabled by bit 8. 0—Border is enabled but not shown; vertical blanking is increased by 8 rows and horizontal blanking is increased by 8 pixels. 1—border is enabled and shown; FRAME_VALID time is extended by 8 rows and LINE_VALID is extended by 8 pixels. See "Pixel Border" on page 124.	0	N	N
Bit 8	Over Sized	0—Normal UXGA size. 1—Adds a 4-pixel border around the active image array independent of readout mode (skip, zoom, mirror, etc.). Setting this bit adds 8 to the number of rows and columns in the frame.	0	Y	YM
Bit 7	Column Skip Enable—Context B	When read mode context B is selected (R0xF2:0[3] = 1): 1—Enable column skip. 0—Normal readout.	0	Y	YM
Bits 6:5	Column Skip—Context B	When read mode context B is selected (R0xF2:0[3] = 1) and column skip is enabled (bit 7 = 1): 00—Column Skip 2x 01—Column Skip 4x 10—Column Skip 8x 11—Column Skip 16x See "Column and Row Skip" on page 125 for more information.	0	Y	YM

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bit 4	Row Skip Enable—Context B	When read mode context B is selected (R0xF2:0[3] = 1): 1—Enable row skip. 0—Normal readout.	0	Y	YM
Bits 3:2	Row Skip—Context B	When read mode context B is selected (R0xF2:0[3] = 1) and Row skip is enabled (bit 4 = 1): 00—Row Skip 2x 01—Row Skip 4x 10—Row Skip 8x 11—Row Skip 16x See "Column and Row Skip" on page 125 for more information.	0	Y	YM
Bit 1	Mirror Columns	Read out columns from right to left (mirrored). When set, column readout starts from column (column start + column size) and continues down to (column start + 1). When clear, readout starts at column start and continues to (column start + column size - 1). This ensures that the starting color is maintained.	0	Y	YM
Bit 0	Mirror Rows	Read out rows from bottom to top (upside down). When set, row readout starts from row (row start + row size) and continues down to (row start + 1). When clear, readout starts at row start and continues to (row start + row size -1). This ensures that the starting color is maintained.	0	Y	YM
R33—0x21 - Read Mode—Context A (R/W)					
Bit 15	Binning—Context A	When read mode context A is selected (R0xF2:0[3] = 0): 0—Normal operation. 1—Binning enabled. See "Binning" on page 127.	1	Y	YM
Bit 10	Use 1 ADC—Context A	When read mode context A is selected (R0xF2:0[3] = 0): 0—Use both ADCs to achieve maximum speed. 1—Use one ADC to reduce power. Maximum readout frequency is now half of the master clock, and the pixel clock is automatically adjusted as described for the pixel clock speed register.	1	Y	YM
Bit 7	Column Skip Enable—Context A	When read mode context A is selected (R0xF2:0[3] = 0): 1—Enable column skip. 0—Normal readout.	1	Y	YM
Bits 6:5	Column Skip—Context A	When read mode context A is selected (R0xF2:0[3] = 0) and column skip is enabled (bit 7 = 1): 00—Column Skip 2x 01—Column Skip 4x 10—Column Skip 8x 11—Column Skip 16x See "Column and Row Skip" on page 125 for more information.	0	Y	YM
Bit 4	Row Skip Enable—Context A	When read mode context A is selected (R0xF2:0[3] = 0): 1—Enable row skip. 0—Normal readout.	1	Y	YM

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bits 3:2	Row Skip—Context A	When read mode context A is selected (R0xF2:0[3] = 0) and Row skip is enabled (bit 4 = 1): 00—Row Skip 2x 01—Row Skip 4x 10—Row Skip 8x 11—Row Skip 16x See "Column and Row Skip" on page 125 for more information.	0	Y	YM
R34—0x22 - Show Control (R/W)					
Bit 10	Number of Dark Columns	The MT9D111 has 40 dark columns. 1—Read out 36 dark columns (4–39). Ignored during binning, where all 40 dark columns are used. 0—Read out 20 dark columns (4–23).	0	N	N
Bit 9	Show Dark Columns	When set, the 20/36 (dependent on bit 10) dark columns are output before the active pixels in a line. There is an idle period of 2 pixels between readout of the dark columns and readout of the active image. Therefore, when set, LINE_VALID is asserted 22 pixel times earlier than normal, and the horizontal blanking time is decreased by the same amount.	0	N	N
Bit 8	Read Dark Columns	1—Enables the readout of dark columns for use in the row-wise noise correction algorithm. The number of columns used are 40 in binning mode, and otherwise determined by bit 10. 0—When disabled, an arbitrary number of dark columns can be read out by including them in the active image. Enabling the dark columns increases the minimum value for horizontal blanking but does not affect the row time.	1	N	Y
Bit 7	Show Dark Rows	When set, the programmed dark rows is output before the active window. FRAME_VALID is thus asserted earlier than normal. This has no effect on integration time or frame rate.	0	N	N
Bits 6:4	Dark Start Address	The start address for the dark rows within the 8 available rows (an offset of 4 is added to compensate for the guard pixels). Must be set so all dark rows read out falls in the address space 0:7.	0	N	N
Bit 3	Reserved	Do not change from default value.			
Bits 2:0	Num Dark Rows	A value of *N* causes (*n* + *1*) dark rows to be read out at the start of each frame when dark row readout is enabled (bit 3).	7	N	Y
R35—0x23 - Flash Control (R/W)					
Bit 15	FLASH	Reflects the current state of FLASH output.	0		
Bit 14	Triggered	Indicates that FLASH output is asserted for the current frame.	0		
Bit 13	Xenon Flash	Enable Xenon flash. When set, FLASH output asserts for the programmed period (bits 7:0) during vertical blanking. This is achieved by keeping the integration time equal to one frame, and the pulse width less than the vertical blank time.	0	Y	N[1]
Bits 12:11	Frame Delay	Delay of the flash pulse measured in frames.	0	N	N

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bit 10	End of Reset	1—In Xenon mode the flash is triggered after the resetting of a frame. 0—In Xenon mode the flash is triggered after the readout of a frame.	1	N	N
Bit 9	Every Frame	1—Flash should be enabled every frame. 0—Flash should be enabled for one frame only.	1	N	N
Bit 8	LED Flash	Enable LED flash. When set, FLASH output asserts prior to the start of the resetting of a frame and remains asserted until the end of the readout of the frame.	0	Y	Y[1]
Bits 7:0	Xenon Count	Length of FLASH pulse when Xenon flash is enabled. The value specifies the length in units of 1024*PIXCLK cycle increments. When the Xenon count is set to its maximum value (0xFF), the FLASH pulse is automatically truncated prior to the readout of the first row, giving the longest pulse possible.	8	N	N
R36—0x24 - Extra Reset (R/W)					
Bit 15	Extra Reset Enable	0—Only programmed window (set by R0x01:0 through R0x04:0) and black pixels are read. 1—Two additional rows are read and reset above and below programmed window to prevent blooming to active area.	1	N	N
Bit 14	Next Row Reset	When set, and the integration time is less than one frame time, row (*n + 1*) is reset immediately prior to resetting row *(n)*. This is intended to prevent blooming across rows under conditions of very high illumination.	0	N	N
Bits 13:0	Reserved	Do not change from default value.			
R37—0x25 - LINE_VALID Control (R/W)					
Bit 15	Xor LINE_VALID	1—LINE_VALID = "continuous" LINE_VALID XOR FRAME_VALID. 0—Normal LINE_VALID (default, no XORing of LINE_VALID). Ineffective if continuous LINE_VALID is set.	0	N	N
Bit 14	Continuous LINE_VALID	1—"Continuous" LINE_VALID (continue producing LINE_VALID during vertical blanking). 0—Normal LINE_VALID (default, no LINE_VALID during vertical blanking).	0	N	N[3]
R38—0x26 - Bottom Dark Rows (R/W)					
Bit 7	Show	The bottom dark rows are visible in the image if the bit is set.	0	N	N
Bits 6:4	Start Address	Defines the start address within the 8 bottom dark rows.	0	N	N
Bit 3	Enable Readout	Enable readout of the bottom dark rows.	0	N	Y
Bits 2:0	Number of Dark Rows	Defines the number of bottom dark rows to be used. (The number of rows used is the specified value + 1.)	7	N	Y
R43—0x2B - Green1 Gain (R/W)					
Bits 11:9	Digital Gain	Total gain = (bit 9 + 1)*(bit 10 + 1)*(bit 11 + 1)*analog gain (each bit gives 2x gain).	0	Y	N
Bits 8:7	Analog Gain	Analog gain = (bit 8 + 1)*(bit 7 + 1)*initial gain (each bit gives 2x gain).	0	Y	N

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bits 6:0	Initial Gain	Initial gain = bits 6:0*0.03125.	20	Y	N
R44—0x2C - Blue Gain (R/W)					
Bits 11:9	Digital Gain	Total gain = (bit 9 + 1)*(bit 10 + 1)*(bit 11 + 1)*analog gain (each bit gives 2x gain).	0	Y	N
Bits 6:0	Initial Gain	Initial gain = bits [6:0]*0.03125.	20	Y	N
Bits 8:7	Analog Gain	Analog gain = (bit 8 + 1)*(bit 7 + 1)*initial gain (each bit gives 2x gain).	0	Y	N
R45—0x2D - Red Gain (R/W)					
Bits 11:9	Digital Gain	Total gain = (bit 9 + 1)*(bit 10] + 1)*(bit 11 + 1)*analog gain (each bit gives 2x gain).	0	Y	N
Bits 8:7	Analog Gain	Analog gain = (bit 8 + 1)*(bit 7 + 1)*initial gain (each bit gives 2x gain).	0	Y	N
Bits 6:0	Initial Gain	Initial gain = bits 6:0*0.03125.	20	Y	N
R46—0x2E - Green2 Gain (R/W)					
Bits 11:9	Digital Gain	Total gain = (bit 9 + 1)*(bit 10 + 1)*(bit 11 + 1)*analog gain (each bit gives 2x gain).	0	Y	N
Bits 8:7	Analog Gain	Analog gain = (bit 8 + 1)*(bit 7 + 1)*initial gain (each bit gives 2x gain).	0	Y	N
Bits 6:0	Initial Gain	Initial gain = bits 6:0*0.03125.	20	Y	N
R47—0x2F - Global Gain (R/W)					
Bits 11:0	Global Gain	This register can be used to simultaneously set all 4 gains. When read, it returns the value stored in R0x2B:0.	20	Y	N
R48—0x30 - Row Noise (R/W)					
Bit 15	Frame-wise Digital Correction	By default, the row noise is calculated and compensated for individually for each color of each row. When this bit is set, the row noise is calculated and applied for each color of each of the first two 2 (two pairs of values) and the same values are applied to each subsequent row, so that new values are calculated and applied once per frame.	0	N	N
Bits 14:12	Gain Threshold	When the upper analog gain bits are equal to or larger than this threshold, the dark column average is used in the row noise correction algorithm. Otherwise, the subtracted value is determined by bit 11. This check is independently performed for each color, and is a means to turn off the black level algorithm for lower gains.	0	N	N
Bit 11	Use Black Level Average	1—Use black level frame average from the dark rows in the row noise correction algorithm for low gains. This frame average was taken before the last adjustment of the offset DAC for that frame, so it might be slightly off. 0—Use mean of black level programmed threshold in the row noise correction algorithm for low gains.	0	N	Y
Bit 10	Enable Correction	1—Enable row noise cancellation algorithm. When this bit is set, the average value of the dark columns read out is used as a correction for the whole row. The dark average is subtracted from each pixel on the row, and then a constant is added (bits 9:0). 0—Normal operation.	1	N	Y

Table 5: Sensor Register Description (continued)

Bit Field		Description	Default (Hex)	Sync'd to Frame Start	Bad Frame
Bits 9:0	Row Noise Constant	Constant used in the row noise cancellation algorithm. It should be set to the dark level targeted by the black level algorithm plus the noise expected between the averaged values of the dark columns. The default constant is set to 42 LSB.	2A	N	Y
R89—0x59 - Black Rows (R/W)					
Bits 7:0	Black Rows	For each bit set, the corresponding dark row (rows 0–7) are used in the black level algorithm. For this to occur, the reading of those rows must be enabled by the settings in R0x22:0.	FF	N	N
R91—0x5B - Green1 Frame Average (R/O)					
Bits 6:0	Green1 Frame Average	The frame-averaged green1 black level that is used in the black level calibration algorithm.			
R92—0x5C - Blue Frame Average (R/O)					
Bits 6:0	Blue Frame Average	The frame-averaged blue black level that is used in the black level calibration algorithm.			
R93—0x5D - Red Frame Average (R/O)					
Bits 6:0	Red Frame Average	The frame-averaged red black level that is used in the black level calibration algorithm.			
R94—0x5E - Green2 Frame Average (R/O)					
Bits 6:0	Green2 Frame Average	The frame-averaged green2 black level that is used in the black level calibration algorithm.			
R95—0x5F - Threshold (R/W)					
Bits 14:8	Upper Threshold	Upper threshold for targeted black level in ADC LSBs.	23	N	N
Bits 6:0	Lower Threshold	Lower threshold for targeted black level in ADC LSBs.	1D	N	N
R96—0x60 - Calibration Control (R/W)					
Bit 15	Disable Rapid Sweep Mode	Disables the rapid sweep mode in the black level algorithm. The averaging mode remains enabled.	0	Y	N
Bit 12	Recalculate	When set, the rapid sweep mode is triggered if enabled, and the running frame average is reset to the current frame average. This bit is write - 1, but always reads back as "0."	0	Y	N
Bit 10	Limit Rapid Sweep	1—Dark rows 8–11 are not used for the black level algorithm controlling the calibration value. Instead, these rows are used to calculate dark averages that can be a starting point for the digital frame-wise black level algorithm. 0—All dark rows can be used for the black level algorithm. This means that the internal average might not correspond to the calibration value used for the frame, so the dark row average should in this case not be used as the starting point for the frame-wise black level algorithm.	0	N	N

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bit 9	Freeze Calibration	When set, does not let the averaging mode of the black level algorithm change the calibration value. Use this with the feature in the frame-wise black level algorithm that allows you to trigger the rapid sweep mode when the dark column average gets away from the black level target.	0	N	N
Bit 8	Sweep Mode	When set, the calibration value is increased by one every frame, and all channels are the same. This can be used to get a ramp input to the ADC from the calibration DACs.	0	N	N
Bits 7:5	Frames To Average Over	Two to the power of this value determines how many frames to average when the black level algorithm is in the averaging mode. In this mode, the running frame average is calculated from the following formula: Running frame ave = old running frame ave (old running frame ave)/2^n + (new frame ave)/ 2^n.	4	N	N
Bit 4	Step Size Forced To 1	When set, the step size is forced to 1 for the rapid sweep algorithm. Default operation (0) is to start at a higher step size when in rapid sweep mode, to converge faster to the correct value.	0	N	N
Bit 3	Switch Calibration Values	When set, the calibration values applied to the two channels are switched. This is not recommended and should not be used.	0		
Bit 2	Same Red/Blue	When this bit is set, the same calibration value is used for red and blue pixels: Calib blue = calib red.	0	N	Y
Bit 1	Same Green	When this bit is set, the same calibration value is used for all green pixels: Calib green2 = calib green1.	0	N	Y
Bit 0	Manual Override	Manual override of black level correction. 1—Override automatic black level correction with programmed values. (R0x61:0–R0x64:0). 0—Normal operation (default).	0	N	Y
R97—0x61 - Green1 Calibration Value (R/W)					
Bits 8:0	Green1 Calibration Value	Analog calibration offset for green1 pixels, represented as a two's complement signed 8-bit value (if bit 8 is clear, the offset is positive and the magnitude is given by bits 7:0. If bit 8 is set, the offset is negative and the magnitude is given by not ([7:0]) + 1). If R0x60:0[0] = 0, this register is R/O and returns the current value computed by the black level calibration algorithm. If R0x60:0[0] = 1, this register is R/W and can be used to set the calibration offset manually. Green1 pixels share rows with red pixels.	0	N	Y
R98—0x62 - Blue Calibration Value (R/W)					
Bits 8:0	Blue Calibration Value	Analog calibration offset for blue pixels, represented as a two's complement signed 8-bit value (if bit 8 is clear, the offset is positive and the magnitude is given by bits 7:0. If bit 8 is set, the offset is negative and the magnitude is given by Not ([7:0]) + 1). If R0x60:0[0] = 0, this register is R/O and returns the current value computed by the black level calibration algorithm. If R0x60:0[0] = 1, this register is R/W and can be used to set the calibration offset manually.	0	N	Y

Table 5: Sensor Register Description (continued)

Bit Field		Description	Default (Hex)	Sync'd to Frame Start	Bad Frame
R99—0x63 - Red Calibration Value (R/W)					
Bits 8:0	Red Calibration Value	Analog calibration offset for red pixels, represented as a two's complement signed 8-bit value (if bit 8 is clear, the offset is positive and the magnitude is given by bits 7:0. If bit 8 is set, the offset is negative and the magnitude is given by Not ([7:0]) + 1). If R0x60:0[0] = 0, this register is R/O and returns the current value computed by the black level calibration algorithm. If R0x60:0[0] = 1, this register is R/W and can be used to manually set the calibration offset.	0	N	Y
R100—0x64 - Green2 Calibration Value (R/W)					
Bits 8:0	Green2 Calibration Value	Analog calibration offset for green2 pixels, represented as a two's complement signed 8-bit value (if bit 8 is clear, the offset is positive and the magnitude is given by bits 7:0. If bit 8 is set, the offset is negative and the magnitude is given by Not ([7:0]) + 1.) If R0x60:0[0] = 0, this register is R/O and returns the current value computed by the black level calibration algorithm. If R0x60:0[0] = 1, this register is R/W and can be used to manually set the calibration offset. Green2 pixels share rows with blue pixels.	0	N	Y
R101—0x65 - Clock (R/W)					
Bit 15	PLL Bypass	1—Bypass the PLL. Use CLKIN input signal as master clock. 0—Use clock produced by PLL as master clock.	1	N	N
Bit 14	PLL Power-down	1—Keep PLL in power-down to save power (default). 0—PLL powered-up.	1	N	N
Bit 13	Power-down PLL During Standby	This register only has an effect when bit 14 = 0. 1—Turn off PLL (power-down) during standby to save power (default). 0—PLL powered-up during standby.	1	N	N
Bit 2	clk_newrow	Force clk_newrow to be on continuously.	0	N	N
Bit 1	clk_newframe	Force clk_newframe to be on continuously.	0	N	N
Bit 0	clk_ship	Force clk_ship to be on continuously.	0	N	N
R102—0x66 - PLL Control 1 (R/W)					
Bits 15:8	M	M value for PLL must be 16 or higher.	28	N	N
Bits 5:0	N	N value for PLL.	9	N	N
R103—0x67 - PLL Control 2 (R/W)					
Bits 11:8	Reserved	Do not change from default value.			
Bits 6:0	P	P value for PLL.	1	N	N
R192—0xC0 - Global Reset Control (R/W)					
Bit 15	Global Reset Enable	Enter global reset. This bit is write - 1 only and is always read 0.	0	N	N[4]
Bit 2	Global Reset Flash Control	1—Flash is de-asserted at end of readout. 0—Flash is de-asserted by R0xB6:0 (de-assert flash).	0	N	N
Bit 1	Global Reset Strobe Control	1—Strobe is de-asserted at end of readout. 0—Strobe is de-asserted by R0xC4:0 (de-assert strobe).	0	N	N

Table 5: Sensor Register Description (continued)

Bit Field	Description		Default (Hex)	Sync'd to Frame Start	Bad Frame
Bit 0	Global Reset Readout Control	1—Start of readout is controlled by falling edge of GRST_CTR. 0—Start of readout is controlled by R0xC2:0 (start readout time).	0	N	N
R193—0xC1 - Start Integration Time (T1) (R/W)					
Bits 15:0	Start Integration Time (T1)	These 16 bits are compared to the upper bits of a 24-bit counter, which starts counting master clocks when global reset starts. When this value is reached, global reset is de-asserted, and integration time starts. There is a minimum time period for which global reset is always held. This time is defined by the physical properties of the boost circuit.	64	N	N
R194—0xC2 (194) Start Readout Time (T2) (R/W)					
Bits 15:0	Start Readout Time (T2)	These 16 bits are added to R0xC1:0 (start integration time) and compared to the 24-bit counter mentioned for R0xC1:0. The value defines the time from when integration time starts to when it is guaranteed to end. Readout then commences.	64	N	N
R195—0xC3 - Assert Strobe Time (T3) (R/W)					
Bits 15:0	Assert Strobe Time (T3)	These 16 bits are compared to the upper bits of a 24-bit counter, which starts counting master clocks when global reset starts. When this value is reached, the strobe is asserted.	96	N	N
R196—0xC4 - De-assert Strobe Time (T4) (R/W)					
Bits 15:0	De-assert Strobe Time (T4)	These 16 bits are compared to the upper bits of a 24-bit counter, which starts counting master clocks when global reset starts. When this value is reached, the strobe is de-asserted if strobe control is "0" (R0xC0:0[1]).	C8	N	N
R197—0xC5 - Assert Flash Time (R/W)					
Bits 15:0	Assert Flash Time	These 16 bits are compared to the upper bits of a 24-bit counter, which starts counting master clocks when global reset starts. When this value is reached, the flash is asserted.	64	N	N
R198—0xC6 - De-assert Flash Time (R/W)					
Bits 15:0	De-assert Flash Time	These 16 bits are compared to the upper bits of a 24-bit counter, which starts counting master clocks when global reset starts. When this value is reached, the flash is de-asserted if flash control is "0" (R0xC0:0[2]).	78	N	N
R224—0xE0 - AIN3 Sample (R/O)					
Bits 9:0	AIN3 Sample	Contains sample of AIN3 if external signal sampling is enabled (R0xE3:0[15] = 1). See "Analog Inputs AIN1–AIN3" on page 134.			
R225—0xE1 - AIN2 Sample (R/O)					
Bits 9:0	AIN2 Sample	Contains sample of AIN2 if external signal sampling is enabled (R0xE3:0[15] = 1).			
R226—0xE2 - AIN1 Sample (R/O)					
Bits 9:0	AIN1 Sample	Contains sample of AIN1 if external signal sampling is enabled (R0xE3:0[15] = 1).			

Table 5: Sensor Register Description (continued)

Bit Field		Description	Default (Hex)	Sync'd to Frame Start	Bad Frame
R227—0xE3 - External Signal Sampling Control (R/W)					
Bit 15	Enable Sampling	1—Enable external signal sampling. 0—Disable external signal sampling.	0	N	N
Bit 14	Show Sample	If external sampling is enabled (R0xE3:0[15] = 1): 1—Show external signal samples in the data stream after LINE_VALID goes LOW. 0—Don't show external signal samples in data stream.	0	N	N
R240—0xF0 - Page Register (R/W)					
Bits 2:0	Page Register	Must be kept at "0" to be able to write/read from sensor. Used in SOC to access other pages with registers.	0	N	N
R241—0xF1 - ByteWise Address (R/W)					
Bits 15:0	Bytewise Address	Special address to perform 16-bit reads and writes to the sensor in 8-bit chunks. See "8-Bit Write Sequence" on page 183.	0	N	N
R242—0xF2 - Context Control (R/W)					
Bit 15	Restart	Setting this bit causes the sensor to abandon the current frame and start resetting the first row. Same physical register as R0x0D:0[1].	0	N	YM
Bit 7	Xenon Flash Enable	Enable Xenon flash. Same physical register as R0x23:0[13].	0	Y	N
Bit 3	Read Mode Select	1—Use read mode context B, R0x20:0. 0—Use read mode context A, R0x21:0. Bits only found in read mode context B register are always taken from that register.	1	Y	YM
Bit 2	LED Flash Enable	Enable LED flash. Same physical register as R0x23:0[8].	0	Y	Y
Bit 1	Vertical Blank Select	1—Use vertical blanking context B, R0x06:0. 0—Use vertical blanking context A, R0x08:0.	1	Y	YM
Bit 0	Horizontal Blank Select	1—Use horizontal blanking context B, R0x05:0. 0—Use horizontal blanking context A, R0x07:0.	1	Y	YM
R255—0xFF - Reserved (R/O)					
Bits 15:0	Reserved	Reserved.	1519		

Note: [1] See "Flash STROBE" on page 131.
[2] Unless integration time is less than one frame.
[3] f enabled by bit 3,
[4] Causes current frame to stop if triggered during a frame.

IFP Registers, Page 1

Table 6: IFP Registers, Page 1

Reg #	Bits	Default	Name
8 0x08	**10:0**	**0x01F8**	**Color Pipeline Control**
	0	0	Toggles the assumption about Bayer CFA (vertical shift). 0—row containing Blue comes first. 1—row with Red comes first.
	1	0	Toggles the assumption about Bayer CFA (horizontal shift). 0—Green comes first. 1—Red or Blue comes first.
	2	0	1 = enable lens shading correction.
	3	1	1 = enable on-the-fly defect correction.
	4	1	1 = enable 2D aperture correction.
	5	1	1 = enable color correction. 0 = bypass color correction (unity color matrix).
	6	1	1 = invert output pixel clock (in all modes - JPEG, SOC, sensor).
	7	1	1 = enable gamma correction.
	8	1	1 = enable decimator.
	9	0	1 = enable minblank. Allows len generation 2 lines earlier. Also adds 4 pixels to the line size.
	10	0	1 = enable 1D aperture correction.
9 0x09	**4:0**	**0x000A**	**Factory Bypass**
	1:0	1	Data output bypass. Selects data going to DOUT pads. 00 = 10-bit sensor. 01 = SOC. 10 = JPEG and output FIFO (no bypass). Reg16 and Reg17 (Page 2) have to be set to the correct output frame size. 11 = Reserved.
	2	0	Reserved.
	4:3	1	GPIO output bypass. 00 = GPIO. 01 = FLASH is output on GPIO11. 10 = Reserved. 11 = Reserved.

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
10 0x0A	**10:0**	**0x0488**	**Pad Slew**
	2:0	0	In bypass mode (R9[1:0] = 2): slew rate for DOUT[7:0], PIXCLK, FRAME_VALID, and LINE_VALID. During normal operation, the slew of listed pads is set by JPEG configuration registers. Actual slew depends on load, temperature, and I/O voltage. Set this register based on customers' characterization results.
	3	1	Unused.
	6:4	0	Slew rate for GPIO[11:0]. Actual slew depends on load, temperature, and I/O voltage. Set this register based on customers' characterization results.
	7	0	1 = enable I/O pad input clamp during standby.That prevents elevated standby current if pad input floating. The pads are GPIO[11:0], DOUT[7:0], FRAME_VALID, LINE_VALID, PIXCLK. 0 = disable I/O pad input clamp. Set this bit to "1" before going to soft/hard standby to reduce the leakage current. When coming out of standby, set this bit back to "0." If none of the GPIOs are used as inputs, this bit can be left at "1."
	10:8	4	Slew rate for SDATA. 7 = fastest slew; 0 = slowest. Actual slew depends on load, temperature, and I/O voltage. Actual slew depends on load, temperature, and I/O voltage. Set this register based on customers' characterization results.
11 0x0B	**8:0**	**0x00DF**	**Internal Clock Control**
	0	1	Reserved.
	1	1	Reserved.
	2	1	Reserved.
	3	1	1 = enable output FIFO clock.
	4	1	1 = enable JPEG clock.
	5	0	Reserved.
	6	1	Reserved.
	7	1	1 = enable GPIO clock.
	8	0	Reserved.
17 0x11	**10:0**	**0x0000**	**X0 Coordinate for Crop Window**
	Use the crop window for pan and zoom. Crop coordinates are updated synchronously with frame enable, unless freeze is enabled. In preview mode crop coordinates are automatically divided by X skip factor. Coordinates are specified as (X0, Y0) and (X1, Y1), where X0 < X1 and Y0 < Y1. Value is overwritten by mode driver (ID = 7).		
18 0x12	**10:0**	**0x0640**	**X1 Coordinate for Crop Window +1**
	See R17:1. Value is overwritten by mode driver (ID = 7).		
19 0x13	**10:0**	**0x0000**	**Y0 Coordinate for Crop Window**
	See R17:1. Value is overwritten by mode driver (ID = 7).		
20 0x14	**10:0**	**0x04B0**	**Y1 Coordinate for Crop Window +1**
	See R17:1. Value is overwritten by mode driver (ID = 7).		

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
21 0x15	**13:0**	**0x0000**	**Decimator Control**
	0	0	Reserved.
	1	0	Reserved.
	2	0	High precision mode. Additional bits for result are stored. Can only be used for decimation > 2.
	3	0	Reserved.
	4	0	Enable 4:2:0 mode.
	5:6	0	Reserved.
	This register controls operation of the decimator. Value is overwritten by mode driver (ID=7).		
22 0x16	**12:0**	**0x0800**	**Weight for Horizontal Decimation** X output = int (X input/2048*reg. value). Value is calculated and overwritten by mode driver (ID = 7).
23 0x17	**12:0**	**0x0800**	**Weight for Vertical Decimation** Y output = int (Y input/2048*reg. value). Value is calculated and overwritten by mode driver (ID = 7).
	InputSize is defined by Registers 17–20. Minimal output size supported by the decimator is 3 x 1 pixel.		
32 0x20	**15:0**	**0xC814**	**Luminance Range of Pixels Considered in WB Statistics**
	7:0	20	Lower limit of luminance for WB statistics.
	15:8	200	Upper limit of luminance for WB statistics.
	In order to avoid skewing WB statistics by very dark or very bright values, this register allows programming the luminance range of pixels to be used for WB computation.		
45 0x2D	**15:0**	**0x5000**	**Right/Left Coordinates of AWB Measurement Window**
	7:0	0	Left window boundary.
	15:8	80	Right window boundary.
	This register specifies the right/left coordinates of the window used by AWB measurement engine. The values programmed in the registers are desired boundaries divided by 8.		
46 0x2E	**15:0**	**0x3C00**	**Bottom/Top Coordinates of AWB Measurement Window**
	7:0	0	Top window boundary.
	15:8	60	Bottom window boundary.
	This register specifies the bottom/top coordinates of the window used by AWB measurement engine. The values programmed in the registers are desired boundaries divided by 8.		
48 0x30	**7:0**	**R/O**	**Red Chrominance Measure Calculated by AWB**
	This register contains a measure of red chrominance obtained using AWB measurement algorithm. The measure is normalized to an arbitrary maximum value; the same for R48:1, R49:1, and R50:1. Because of this normalization, only the ratios of values of registers R48:1 R49:1, and R50:1 should be used.		
49 0x31	**7:0**	**R/O**	**Luminance Measure Calculated by AWB**
	This register contains a measure of image luminance obtained using AWB measurement algorithm. The measure is normalized to an arbitrary maximum value; the same for R48:1, R49:1, and R50:1. Because of this normalization, only the ratios of values of registers R48:1, R49:1, and R50:1 should be used.		
50 0x32	**7:0**	**R/O**	**Blue Chrominance Measure Calculated by AWB**
	This register contains a measure of blue chrominance obtained using AWB measurement algorithm. The measure is normalized to an arbitrary maximum value; the same for R48:1, R49:1, and R50:1. Because of this normalization, only the ratios of values of registers R48:1, R49:1, and R50:1 should be used.		
53 0x35	**15:0**	**0x1208**	**1D Aperture Correction Parameters**
	7:0	8	Ap_knee; threshold for aperture signal.
	10:8	2	Ap_gain; gain for aperture signal.
	13:11	2	Ap_exp; exponent for gain for aperture signal.

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
54 0x36	**15:0**	**0x1208**	**2D Aperture Correction Parameters**
	7:0	8	Ap_knee; threshold for aperture signal.
	10:8	2	Ap_gain; gain for aperture signal.
	13:11	2	Ap_exp; exponent for gain for aperture signal.
	14	0	Reserved.
	Defines 2D aperture gain and threshold.		
55 0x37	**7:0**	**0x080**	**Filters**
	2:0	0	UV filter: 000—no filter 001—11110 averaging filter 010—01100 averaging filter 011—01210 averaging filter 100—12221 averaging filter 101—median 3 110—median 5 111—Reserved
	4:3	0	Y filter mode: 00—no filter 01—median 3 10—median 5 11—Reserved
	5	0	Permanently enable Y filter.
59 0x3B	**9:0**	**0**	**Second Black Level**
	This register contains the value subtracted by IFP from pixel values prior to CCM. If the subtraction produces a negative result for a particular pixel, the value of this pixel is set to "0."		
60 0x3C	**9:0**	**42**	**First Black Level**
	This register contains the value subtracted by IFP from raw pixel values before applying lens shading correction and digital gains. If the subtraction produces a negative result for a particular pixel, the value of this pixel is set to "0." Typically, the subtracted value should be equal to the black level targeted by the sensor. This value is subtracted from all test patterns as well.		
67 0x43	**0**	**1**	**Enable Support for Preview Modes**
	1- enable. Enable automatic recalculation operation of coefficient lens correction dependent on sensor output resolution.		
68 0x44	**15:0**	**N/A**	**Mirrors Sensor Register 0x20**
	Read only		
69 0x45	**15:0**	**N/A**	**Mirrors Sensor Register 0xF2**
	Read only		
70 0x46	**15:0**	**N/A**	**Mirrors Sensor Register 0x21**
	Read only		
71 0x47	**7:0**	**16**	**Edge Threshold for Interpolation**
	Threshold for identifying pixel neighborhood as having an edge.		

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
72 0x48	**2:0**	**0**	**Test Pattern**
	2:0	0	Test mode. 001—flat field; RGB values are specified in R73-75:1 010—vertical monochrome ramp 011—vertical color bars 100—vertical monochrome bars; set bar intensity in R73:1 and R74:1 101—pseudo-random test pattern 110—horizontal monochrome bars; set bar intensity in R73:1 and R74:1
	Injects test pattern into beginning of color pipeline.		
73 0x49	**9:0**	**256**	**Test Pattern R/Monochrome Value**
74 0x4A	**9:0**	**256**	**Test Pattern G/Monochrome Value**
75 0x4B	**9:0**	**256**	**Test Pattern B Value**
78 0x4E	**7:0**	**32**	**Digital Gain 2**
	Default setting 32 corresponds to gain value of 1. Gain scales linearly with value.		
96 0x60	**14:0**	**0x2923**	**Color Correction Matrix Exponents for C11..C22** 2:0—matrix element 1 (C11) exponent 5:3—matrix element 2 (C12) exponent 8:6—matrix element 3 (C13) exponent 11:9—matrix element 4 (C21) exponent 14:12—matrix element 5 (C22) exponent
97 0x61	**11:0**	**0x0524**	**Color Correction Matrix Exponents for C22..C33** 2:0—matrix element 6 (C23) exponent 5:3—matrix element 7(C31) exponent 8:6—matrix element 8(C32) exponent 11:9— matrix element 9(C33) exponent The value of a matrix coefficient is calculated Cij=(1 - 2*Sij)*Mij*2^-(Eij + 4), 0< = Eij < = 4 Here Sij is coefficient's sign, Mij is the mantissa and Eij is the exponent.
98 0x62	**15:0**	**0xBBC8**	**Color Correction Matrix Elements 1 and 2 Mantissas**
	7:0	200	Color correction matrix element 1 (C11).
	15:8	187	Color correction matrix element 2 (C12).
99 0x63	**15:0**	**0xCA3A**	**Color Correction Matrix Elements 3 and 4 Mantissas**
	7:0	58	Color correction matrix element 3 (C13).
	15:8	202	Color correction matrix element 4 (C21).
100 0x64	**15:0**	**0x3B85**	**Color Correction Matrix Elements 5 and 6 Mantissas**
	7:0	133	Color correction matrix element 5 (C22).
	15:8	59	Color correction matrix element 6 (C23).
101 0x65	**15:0**	**0xF26F**	**Color Correction Matrix Elements 7 and 8 Mantissas**
	7:0	111	Color correction matrix element 7 (C31).
	15:8	242	Color correction matrix element 8 (C32).

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
102 0x66	13:0	0x3D9C	**Color Correction Matrix Element 9 Mantissa and Signs**
	7:0	156	Color correction matrix element 9 (C33).
	13:8	61	Signs for off-diagonal CCM elements. Bit 8—sign for C12 Bit 9—sign for C13 Bit 10—sign for C21 Bit 11—sign for C23 Bit 12—sign for C31 Bit 13—sign for C32 1— indicates negative 0—indicates positive Signs for C11, C22, and C33 are assumed always positive.
106 0x6A	9:0	128	**Digital Gain 1 for Red Pixels**
	Default setting 128 corresponds to gain value of 1. Gain scales linearly with value.		
107 0x6B	9:0	128	**Digital Gain 1 for Green1 Pixels**
	Default setting 128 corresponds to gain value of 1. Gain scales linearly with value.		
108 0x6C	9:0	128	**Digital Gain 1 for Green2 Pixels**
	Default setting 128 corresponds to gain value of 1. Gain scales linearly with value.		
109 0x6D	9:0	128	**Digital Gain 1 for Blue Pixels**
	Default setting 128 corresponds to gain value of 1. Gain scales linearly with value.		
110 0x6E	9:0	128	**Digital Gain 1 for All Colors**
	Write 128 to set all gains (R106-R109) to 1. When read, this register returns the value of R107.		
122 0x7A	15:0	80	**Boundaries of Flicker Measurement Window (Left/Width)**
	7:0	80	Window width.
	15:8	0	Left window boundary.
	This register specifies the boundaries of the window used by the flicker measurement engine. The values programmed in the registers are the desired boundaries divided by 8.		
123 0x7B	15:0	0x0088	**Boundaries of Flicker Measurement Window (Top/Height)**
	5:0	8	Window height.
	15:6	2	Top window boundary.
	This register specifies the boundaries of the window used by the flicker measurement engine.		
124 0x7C	15:0	5120	**Flicker Measurement Window Size**
	This register specifies the number of pixels in the window used by the flicker measurement engine.		
125 0x7D	15:0	R/O	**Measure of Average Luminance in Flicker Measurement Window**
150 0x96	0	0	**Blank Frames**
	0	0	1 = blank outgoing frames. When set, current frame completes and following frames are not output, FEN = LEN = 0. When unset, output resumes with the next frame. The bit is synchronized with frame enable. See freeze bit (R152[7]).

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
151 0x97	**7:0**	**0**	**Output Format Configuration**
	0	0	In YUV output mode, swaps Cb and Cr channels. In RGB mode, swaps R and B. This bit is subject to synchronous update.
	1	0	Swaps chrominance byte with luminance byte in YUV output. In RGB mode, swaps odd and even bytes. This bit is subject to synchronous update.
	2	0	Reserved.
	3	0	Monochrome output.
	4	0	1 = use ITU-R BT.601 codes when bypassing FIFO. (0xAB = frame start; 0x80 = line start; 0x9D = line end; 0xB6 = frame end.)
	5	0	1 = output RGB (see R151[7:6]) 0 = output YUV
	7:6	0	RGB output format: 00 = 16-bit RGB565 01 = 15-bit RGB555 10 = 12-bit RGB444x 11 = 12-bit RGBx444
152 0x98	**2:0**	**0**	**Output Format Test**
	0	0	1 = disable Cr channel (R in RGB mode).
	1	0	1 = disable Y channel (G in RGB mode).
	2	0	1 = disable Cb channel (B in RGB mode).
	5:3	0	Test ramp output: 00—off 01—by column 10—by row 11—by frame
	6	0	1 = enable 8+2 bypass.
	7	0	1 = freeze update SOC registers affecting output size and format. These include R151:1, R17-23:1.
153 0x99	**11:0**	**R/O**	**Line Count**
154 0x9A	**15:0**	**R/O**	**Frame Count**
164 0xA4	**15:0**	**0x6440**	**Special Effects**
	2:0	0	Special effect selection bits: 0—disabled 1—monochrome 2—sepia 3—negative 4—solarization with unmodified UV 5 —solarization with -UV
	5:3	0	Dither enable and amplitude. Valid values are 1..4, others disable.
	6	1	1 = dither only in luma channel, 0 = dither in all color channels.
	15:8	0	Solarization threshold for luma, divided by 2; 64..127.
165 0xA5	**15:0**	**0xB023**	**Sepia Constants**
	7:0	35	Sepia constant for Cr.
	15:8	176	Sepia constant for Cb.

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
178 0xB2	**15:0**	**0x2700**	**Gamma Curve Knees 0 and 1**
	7:0		Ordinate of gamma curve knee point 0 (its abscissa is 0).
	15:8		Gamma curve knee point 1.
	Gamma correction curve is implemented in the MT9D111 as a piecewise linear function mapping 12-bit arguments to 8-bit output. Seventeen line fragments forming the curve connect 19 knee points, whose abscissas are fixed and ordinates are programmable through registers R[178:187]:1. The abscissas of the knee points are 0, 64, 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2304, 2560, 2816, 3072, 3328, 3584, 3840, and 4095. Ordinates of knee points written directly to the registers R[178:187]:1 may be overwritten by mode driver (ID=7). The recommended way to program a gamma curve into the MT9D111 is to write desired knee ordinates to one of the two mode.gamma_table[A/B][] arrays that hold gamma curves used in contexts A and B. Once the desired ordinates are in one of those arrays, all that is needed to put them into effect (i.e. into the registers) is a short command telling the sequencer to go to proper context or through REFRESH loop.		
179 0xB3	**15:0**	**0x4936**	**Gamma Curve Knees 2 and 3**
	7:0		Gamma curve knee point 2.
	15:8		Gamma curve knee point 3.
180 0xB4	**15:0**	**0x7864**	**Gamma Curve Knees 4 and 5**
	7:0		Gamma curve knee point 4.
	15:8		Gamma curve knee point 5.
181 0xB5	**15:0**	**0x9789**	**Gamma Curve Knees 6 and 7**
	7:0		Gamma curve knee point 6.
	15:8		Gamma curve knee point 7.
182 0xB6	**15:0**	**0xB0A4**	**Gamma Curve Knees 8 and 9**
	7:0		**Gamma curve knee point 8.**
	15:8		Gamma curve knee point 9.
183 0xB7	**15:0**	**0xC5BB**	**Gamma Curve Knees 10 and 11**
	7:0		Gamma curve knee point 10.
	15:8		Gamma curve knee point 11.
184 0xB8	**15:0**	**0xD7CE**	**Gamma Curve Knees 12 and 13**
	7:0		Gamma curve knee point 12.
	15:8		Gamma curve knee point 13.
185 0xB9	**15:0**	**0xE8E0**	**Gamma Curve Knees 14 and 15**
	7:0		Gamma curve knee point 14.
	15:8		Gamma curve knee point 15.
186 0xBA	**15:0**	**0xF8F0**	**Gamma Curve Knees 16 and 17**
	7:0		Gamma curve knee point 16.
	15:8		Gamma curve knee point 17.
187 0xBB	**7:0**	**0x00FF**	**Gamma Curve Knee 18**
190 0xBE	**3:0**	**6**	**YUV/YCbCr control**
	3	0	Clips Y values to 16–235; clips UV values to 16–240.
	2	1	Adds 128 to U and V values.
	1	1	1 = ITU-R BT.601 coefficients 0 = use sRGB coefficients.
	0	0	0 = no scaling 1 = scales Y data by 219/256 and UV data by 224/256.

Table 6: IFP Registers, Page 1 (continued)

Reg #	Bits	Default	Name
191 0xBF	**15:0**	**0**	**Y/RGB Offset**
	15:8	0	Y offset.
	7:0	0	RGB offset.
195 0xC3	**15:0**	**0**	**Microcontroller Boot Mode**
	0	0	1 = reset microcontroller.
	1,2,3,6:4, 6:5,7:5	0	Reserved.
	7	0	Microcontroller debug indicator.
	11:8,12,13, 14,15	R/O	Reserved.
198 0xC6	**15:0**	**0**	**Microcontroller Variable Address**
	7:0	0	Bits 7:0 of address for physical access; driver variable offset for logical access.
	12:8	0	Bits 12:8 of address for physical access; driver ID for logical access.
	14:13	0	Bits 14:13 of address for physical access; R198:1[14:13] = 01 select logical access.
	15	0	1 = 8-bit access; 0 = 16-bit.
	Microcontroller variables are similar to two-wire serial interface registers, except that they are located in the microcontroller's memory and have different bit widths. Registers 198:1 and R200:1 provide easy access to variables that can be represented as 8- or 16-bit unsigned integers (bytes or words). To access such a variable, one must write its address to R198:1 and then read its value from R200:1 or write a new value to the same register. Variables having more than 16 bits (e.g. 32-bit unsigned long integers) must be accessed as arrays of bytes or words - there is no way to read or write their values without parsing. Variable address written to R198:1 can be physical or logical. Physical address is the actual address of a byte or word in the microcontroller's address space. Physical addresses can be used in many manipulations of memory content, for example, to upload custom binary code to a specific RAM segment. The logical addressing option is provided only to facilitate access to public variables of various firmware drivers. Logical address of a public variable consists of a 5-bit driver ID (0 = monitor, 1 = sequencer, etc.) and 8-bit offset of the variable in the driver's data structure (which cannot be larger than 256 bytes).		
X4 0xC8	**15:0**	**0**	**Microcontroller Variable Data**
	To read current value of an 8- or 16-bit variable from microcontroller memory, write its address to R198:1 and read R200:1. To change value of a variable, write its address to R198:1 and its new value to R200:1. When bit width of a variable is specified as 8 bits (R198:1[15]=1), the upper byte of R200:1 is irrelevant. It is set to "0" when the register is read and ignored when it is written to. See R198:1 above for explanation how to read and set variables having more than 16 bits.		
201-209 0xC9-D1	**15:0**	**0**	**Microcontroller Variable Data using Burst Two-Wire Serial Interface Access**
	Use these registers to read or write up to 16 bytes of variable data using the burst two-wire serial interface access mode. The variables must have consecutive addresses.		
240 0xF0	**2:0**	**1**	**Page Register** 0 = sensor core 1 = IFP page 1 2 = IFP page 2
241 0xF1	**15:0**	**0**	**Bytewise Address** Special address to perform 16-bit READs and WRITEs to the sensor in 8-bit chunks. See "8-Bit Write Sequence" on page 183.

IFP Registers, Page 2

Table 7: IFP Registers, Page 2

Reg #	Bits	Default	Name
0 0x00	**15:0**	**0**	**JPEG Control Register**
	0	0	Start/Enable Encoder: Enable JPEG encoding at the start of next frame.
	1	0	Test SRAM: When set, allows host or microcontroller to take control of the output FIFO buffer and the sixteen 800 x 16 RAMs in the re-order buffer for testing. Used in conjunction with JPEG RAM test controls register to simultaneously write all 17 RAMs and individually read each RAM.
	[14:2]		Return zero when read.
	15	0	Soft Reset.
2 0x02	**15:0**	**0**	**JPEG Status Register 0**
	0	0	Transfer done status flag. When asserted, it indicates that the completion of transfer of the JPEG-compressed image. This status flag remains set until cleared by the host or microcontroller by writing "1" to bit [0]. Subsequently, the output FIFO overflow, the spoof oversize error and the re-order buffer error status bits are reset. The output buffer clock must be present to clear this bit.
	1	0	Output FIFO overflow status flag. When asserted, it indicates that an overflow condition was detected in the output FIFO during the frame transfer and that transfer was terminated prematurely. This status flag remains set until cleared by the host or microcontroller as it clears transfer done flag. Valid for JPEG compressed images only.
	2	0	Spoof oversize error status flag. When asserted, it indicates that the spoof frame size is too small for JPEG data stream. This status flag remains set until cleared by the host or microcontroller as it clears transfer done flag. Valid for JPEG compressed images only.
	3	0	Re-order buffer error status flag: When the re-order buffer detects an overflow or underflow condition, this bit is set to "1." This bit is cleared by writing a "1" to bit[0] of this register.
	5:4	0	Watermark of the output FIFO. 00—less than 25% full 01—25% to less than 50% full 10—50% to less than 75% full 11—75% full or more Watermark is cleared when the host or microcontroller writes "1" to bit 4 of this register (R258).
	7:6	0	QTable_ID. 00—Quantization table set 0 01—Quantization table set 1 10—Quantization table set 2 11—Reserved
	15:8	0	JPEG data length bits 23:16. Highest byte of 24-bit JPEG data length.
3 0x03	**15:0**	**0**	**JPEG Status Register 1 - JPEG Data Length Bits 15:0**
	This register combined with R2:2[15:8] gives the total number of data bytes successfully encoded—a 24-bit JPEG data length. If an output FIFO overflow occurs, this register holds the total number of data bytes already sent out by the JPEG encoder (up to the point where the overflow occurs).		
4 0x04	**2:0**	**0**	**JPEG Status Register 2 - Output FIFO Fullness Status**
	Instantaneous FIFO fullness status code: 000—FIFO is empty 001—0% < fullness < 25% 011—25% <= fullness < 50% 010—50% <= fullness < 75% 110—75% <= fullness <= 100%		

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
5 0x05	**5:0**	**0**	**JPEG Front End Configuration Register**
	0	0	Color component composition: 0—4:2:2 format 1—4:2:0 format
	1	0	JPEG monochrome mode. When this bit is set, the re-order buffer control sends only luma data to the JPEG encoder.
6 0x06	**0:0**	**0**	**JPEG Core Configuration Register - Extend JPEG Quantization Matrix**
	Presently, the JPEG encoder core supports two sets of quantization tables. (Each set contains a pair of quantization tables - one for luma, one for chroma.) The quantization memory available allows an additional set of quantization tables to be stored. By setting this bit, the encoder uses the third table pair. If reset, then whichever set of tables 1 and 2 was programmed.		
10 0x0A	**0:0**	**1**	**JPEG Encoder Bypass**
	Set bit 0 of this register to have uncompressed frames from the SOC captured in the output FIFO and transferred out as spoof frames (JPEG encoder is bypassed). Register 13, bit 0, should be set to “1” for spoof-mode output. When the bit is cleared, JPEG-encoded frames are transferred out through the FIFO.		

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
13 0x0D	**10:0**	**0x0007**	**Output Configuration Register**
	0	1	Enable spoof frame: When this bit is set, the data captured in the output FIFO is sent out as a spoofed frame, formatted according to information stored in the various spoof registers. This output mode can be used to output both JPEG-compressed and uncompressed image data. JPEG data may be padded if dummy data is needed. During LINE_VALID assertion period, the output clock is only clocked when there is valid JPEG data or padded dummy data to be transmitted. This may result in a non-uniform clock period. When LINE_VALID is de-asserted, the output clock is enabled according to SPOOF_LV_LEAD and SPOOF_LV_TRAIL setting. JPEG SOI/EOI markers cannot be inserted into spoof frames.
	1	1	Enable output pixel clock between frames: When set, this bit enables the pixel clock to run whether FRAME_VALID is LOW or HIGH. Clearing the bit disables the clock during the periods when FRAME_VALID is de-asserted except for SOI/EOI markers transmission if they are outside the FV assertion period. The gating off of this output pixel clock saves power.
	2	1	Enable pixel clock during invalid data: When this bit is set, the pixel clock runs continuously while FRAME_VALID is asserted but LINE_VALID is de-asserted. When cleared, it causes the pixel clock to be active only when valid JPEG data are output. Disabling pixel clock during LINE_VALID de-assertion is not support in spoof frame.
	3	0	Enable SOI/EOI insertion: When set, this bit causes SOI and EOI markers to be output at the beginning and end of every JPEG-encoded frame. When the bit is cleared, only JPEG data bytes are output.
	4	0	Insert SOI and EOI when FRAME_VALID is HIGH: When it is “1,” SOI and EOI are inside FV assertion period. When it is“0,” SOI, and EOI are outside FRAME_VALID assertion period. This bit is relevant only when SOI/EOI insertion is enabled.
	5	0	Ignore spoof frame height: This bit is used in conjunction with bit 0 that enables spoof framing. When this bit is set, the JPEG unit output section ignores the spoof frame height register. The spoof frame ends when either JPEG bytes or uncompressed image data are exhausted. Both kinds of data are always padded with dummy data to the programmed spoof frame width.
	6	0	Enable variable pixel clock rate: Setting this bit enables the adaptive pixel clock frequency feature. The pixel clock is switched from PCLK1 to PCLK2 to PCLK3, depending on the output FIFO fullness. When fullness reaches 50%, the pixel clock is switched from PCLK1 to PCLK2. When the fullness reaches 75%, the clock is switched to PCLK3. As the FIFO fullness drops below 50%, PIXCLK switches from PCLK3 to PCLK2; as it drops below 25%, it switches back to PCLK1. At the start of a frame, it always starts with PCLK1.
	7	0	Enable byte swap: Toggling this bit swaps the even and odd bytes in JPEG data stream. Byte swapping supported only when enable spoof frame is set.

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
	8	0	Duplicate FRAME_VALID on LINE_VALID: When this bit is set, the FRAME_VALID waveform is output on LINE_VALID output also; therefore, the two are identical.
	9	0	Enable status insertion: When this bit is set, the JPEG module appends the status byte to the end of the JPEG byte stream. This register should only be set when transferring in continuous mode. The status byte inserted at the end of the JPEG byte stream is Reg2 bit [7:0].
	10	0	Enable spoof ITU-R BT.601 codes: This bit is relevant matters when uncompressed frames are output in the spoof mode. When set, the bit causes the ITU-R BT.601 markers SOF, EOF, SOL, EOL to be inserted into every frame. Codes are: Start of Frame: FF0000AB End of Frame: FF0000B6 Start of Line: FF000080 End of Line: FF00009D
	11	0	Freeze_update; Default = 0. When set, it disables the transfer of values from registers 0x0A, 0x0D (except freeze_update), 0x0E, 0x0F, 0x10, 0x11, 0x12 into their corresponding shadow registers. When cleared, the shadow registers are updated with values from their corresponding configuration registers at the end of vertical blanking of the input image. The shadow registers allow the microcontroller to change configuration registers values during the active frame time without corrupting the current frame transfer.
14 0x0E	**15:0**	**0x0501**	**Output PCLK1 & PCLK2 Configuration Register**
	3:0	0x1	Output clock frequency divisor N1: This 4-bit register contains an integer divisor used to divide master clock frequency to obtain the frequency of output clock source PCLK1. A value of "0" in this register has the same effect as "1."
	7:5	0	PCLK1 slew rate: The value contained in this 3-bit register determines the slew rate of the output clock when PCLK1 is selected as its source.
	11:8	0x5	Output Clock Frequency Divisor N2: This 4-bit register contains an integer divisor used to divide master clock frequency to obtain the frequency of output clock source PCLK2. The output clock switches from PCLK1 to PCLK2 when the output buffer fullness reaches 50%. A value of "0" in this register has the same effect as "1."
	12	0	Not used.
	15:13	0	PCLK2 slew rate: The value contained in this 3-bit register determines the slew rate of the output clock when PCLK2 is selected as its source.
15 0x0F	**7:0**	**0x0003**	**Output PCLK3 Configuration Register**
	3:0	0x03	Output clock frequency divisor N3: This 4-bit register contains an integer divisor used to divide master clock frequency to obtain the frequency of output clock source PCLK3. The output clock switches from PCLK2 to PCLK3 when the output buffer fullness reaches 75 %. A value of "0" in this register has the same effect as 1.
	4	0	Not used.
	7:5	0	PCLK3 slew rate: The value contained in this 3-bit register determines the slew rate of the output clock when PCLK3 is selected as its source.
16 0x10	**11:0**	**0x0640**	**Spoof Frame Width**
	This register defines the width of the spoof frame used to output data captured in the output FIFO. It corresponds to the number of valid data bytes output at each assertion of LINE_VALID. It must be an even number.		

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
17 0x11	**11:0**	**0x0258**	**Spoof Frame Height**
	This register defines the height of the spoof frame used to output data captured in the output FIFO. The height is equal to the number of assertions of LINE_VALID within one assertion of FRAME_VALID. The value stored in this register is ignored if bit R13:2[5] is set.		
18 0x12	**15:0**	**0x0606**	**Spoof Frame Line Timing**
	7:0	0x6	Spoof LINE_VALID lead. The number of clocks before LINE_VALID is asserted in a spoof frame. This must be a minimum value of "5."
	15:8	0x6	Spoof LINE_VALID trail. The number of clocks after LINE_VALID is de-asserted in a spoof frame. This must be a minimum value of "5."
29 0x1D	**15:0**	**0**	**JPEG RAM Test Control Register**
	9:0	0	Tested SRAM address: This register defines the location in the seventeen 800 x 16 RAMs that is being accessed by the host or microcontroller while testing the group of SRAMs. A specified 16-bit location can be selected from 0 through 799. The address is automatically incremented when R31:2 is written during SRAM test data Write or read during SRAM test data read.
	12:10	0	Test Y/C SRAM Register: Address the same set of 8 SRAMs (either for Y or for C), this register setting identifies which of the 8 SRAMs is selected. 000 for SRAM1, 001 for SRAM2, …, 111 for SRAM8.
	13	0	Test Y/C SRAM Select Register: Select a bank of 8 SRAMs from luminance or from chrominance. Y = 1, C = 0.
	14	0	Test output buffer SRAM: Select to read output buffer SRAM and supersedes Y/C SRAMs. If set, data from the output buffer RAM is selected to be read. During data WRITE, this has no effect.
	15	0	SRAM write enable. This bit is used in conjunction with the RAM selection register. When this bit is set, the RAM specified in the selection register undergoes a test write cycle. Data residing in the indirect data register R31:2 is loaded into all seventeen 800 x 16 SRAMS simultaneously. Resetting the bit thereafter causes a test read cycle to be performed and the data is read from the SRAMs and loaded back into R31:2. This bit is write_enable when 1; read_enable when 0. The READ and WRITE cycles occur when R31:2 is accessed.
30 0x1E	**15:0**	**0**	**JPEG Indirect Access Control Register**
	10:0	0	Indirect access address register: This 11-bit register contains the address of the register or memory to be accessed indirectly.
	12:11	0	Unused.
	13	0	Enable two-wire serial interface burst: When this bit is set, the two-wire serial interface decoder operates in burst mode for the indirect data register (READ burst and WRITE burst). The longest burst supported is 16 (128 READ or WRITE cycles).
	14	0	Enable indirect writing: When set, data from the indirect data register is written to the Indirect address location specified by [10:0] of this register except when auto-increment is set. Reading the same address location when this bit is reset to "0."
	15	0	Address auto-increment: When this bit is set, the value in the indirect access address register is automatically incremented after every read or write, to the JPEG indirect access data register. This feature is used to emulate a burst access to memory or registers being accessed indirectly.
31 0x1F	**15:0**	**0**	**JPEG Indirect Access Data Register**
	Writing to, and reading from this register, is equivalent to performing these operations on registers or memory being indirectly accessed. When address auto-increment bit is set in JPEG indirect access control register, multiple writes or reads from this register affect a burst data transfer. Data is written to or read from Indirect registers (when TestSRAM REG 0x0[1] is set to "0"), or the 800 x 16 SRAMs (when TestSRAM is set to "1").		

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
32-46 0x20-0x2E	**15:0**	0	**JPEG Indirect Access Data Register** Same as R31:2 when doing two-wire serial interface burst.
64 0x40	**15:0**	**0x0F14**	**Boundaries of First AF Measurement Window (Top/Left)**
	7:0		Left window boundary.
	15:8		Top window boundary.
	This register specifies top and left boundaries of the first from 16 (left-top) window used by AF measurement engine. The values programmed in the registers are desired boundaries divided by 8.		
65 0x41	**15:0**	**0x1E28**	**Boundaries of First AF Measurement Window (Height/Width)**
	7:0		Window width.
	15:8		Window height.
	This register specifies height and width of the first from 16 window used by AF measurement engine. The values programmed in the registers are desired boundaries divided by 2.		
66 0x42	**15:0**	**0x12C0**	**AF Measurement Window Size**
	This register specifies number of pixels in the window used by AF measurement engine.		
67 0x43	**15:0**	**R/O**	**Average Luminance in AF Windows W12 and W11**
	7:0		Average Y in W11.
	15:8		Average Y in W12.
68 0x44	**15:0**	**R/O**	**Average Luminance in AF Windows W14 and W13**
	7:0		Average Y in W13.
	15:8		Average Y in W14.
69 0x45	**15:0**	**R/O**	**Average Luminance in AF Windows W22 and W21**
	7:0		Average Y in W21.
	15:8		Average Y in W21.
70 0x46	**15:0**	**R/O**	**Average Luminance in AF Windows W24 and W23**
	7:0		Average Y in W23.
	15:8		Average Y in W24.
71 0x47	**15:0**	**R/O**	**Average Luminance in AF Windows W32 and W31**
	7:0		Average Y in W31.
	15:8		Average Y in W32.
72 0x48	**15:0**	**R/O**	**Average Luminance in AF Windows W34 and W33**
	7:0		Average Y in W33.
	15:8		Average Y in W34.
73 0x49	**15:0**	**R/O**	**Average Luminance in AF Windows W42 and W41**
	7:0		Average Y in W41.
	15:8		Average Y in W43.
74 0x4A	**15:0**	**R/O**	**Average Luminance in AF Windows W44 and W43**
	7:0		Average Y in W43.
	15:8		Average Y in W44.

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
75 0x4B	**15:0**	**0x028**	**AF Filter 1 Coefficients**
	3:0	1	C1.
	7:4	0	C2.
	11:8	0	C3.
	15:12	0	C4.
	This register specifies coefficient pairs for horizontal filter 1. Coefficient pairs C1 - C4 and center coefficient C0 define the filter kernel. Filter tap order is C4, C3, C2, C1, C0,C 1, C2, C3, C4 for odd-length configuration and C4, C3, C2, C1, C1, C2, C3, C4 for even-sized. R76:2[5] controls filter symmetry. When "0" filter the even-length configuration becomes C4, C3, C2, C1, -C1, -C2, -C3, -C4. The filter input is linear luminance from interpolation. Filter coefficients are specified as unsigned 4-bit numbers, 0..15. Their signs are specified in R76:2.		
76 0x4C	**15:0**	**0xC0B0**	**AF Filter 1 Configuration**
	3:0	0	Scale factor 0- 1; 1- /2; 2- /4; 3- /8; 4- /16; 5- /32; 6- /64; 7- /128; 8- /256; 9- /512; 11- *2; 12- *4; 13- *8; 14- *16; 15- *32.
	4	1	1 = odd length (9 taps); 0 = even length (8 taps, C0 ignored).
	5	1	If "1" filter is symmetric else anti-symmetric.
	10:6	0	Signs for filter coefficients C4(10):C0(6). 1 = negative, 0 = positive.
	15:12	0	C0 coefficient, for the center tap in odd-sized filter.
	If filter coefficients larger than 1, the user must use appropriate scale factor to avoid overflow. For example, use scale factor /2 for filter 0 0 0 2 -2 0 0 0.		
77 0x4D	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W12 and W11**
	7:0		Average sharpness in W11 (top left).
	15:8		Average sharpness in W12.
78 0x4E	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W14 and W13**
	7:0		Average sharpness in W13.
	15:8		Average sharpness in W14 (top right).
79 0x4F	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W22 and W21**
	7:0		Average sharpness in W21.
	15:8		Average sharpness in W22.
80 0x50	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W24 and W23**
	7:0		Average sharpness in W23.
	15:8		Average sharpness in W24.
81 0x51	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W32 and W31**
	7:0		Average sharpness in W31.
	15:8		Average sharpness in W32.
82 0x52	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W34 and W33**
	7:0		Average sharpness in W33.
	15:8		Average sharpness in W34.
83 0x53	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W42 and W41**
	7:0		Average sharpness in W41 (bottom left).
	15:8		Average sharpness in W42.
84 0x54	**15:0**	**R/O**	**AF Filter 1 Average Sharpness Measure for AF Windows W44 and W43**
	7:0		Average sharpness in W43.
	15:8		Average sharpness in W44 (bottom right).

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
85 0x55	**15:0**	**0x2080**	**AF Filter 2 Coefficients**
	3:0	1	C1.
	7:4	1	C2.
	11:8	1	C3.
	15:12	1	C4.
	See R75:2.		
86 0x56	**15:0**	**0xC130**	**AF Filter 2 Configuration**
	3:0	0	Scale factor.
	4	1	Odd/even (9/8) filter size.
	5	1	If "1" filter is symmetric else anti-symmetric.
	10:6	0	Signs for filter coefficients C4(10):C0(6).
	15:12	0	C0 coefficient.
	See R76:2.		
87 0x57	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W12 and W11**
	7:0		Average sharpness in W11.
	15:8		Average sharpness in W12.
88 0x58	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W14 and W13**
	7:0		Average sharpness in W13.
	15:8		Average sharpness in W14.
89 0x59	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W22 and W21**
	7:0		Average sharpness in W21.
	15:8		Average sharpness in W22.
90 0x5A	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W24 and W23**
	7:0		Average sharpness in W23.
	15:8		Average sharpness in W24.
91 0x5B	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W32 and W31**
	7:0		Average sharpness in W31.
	15:8		Average sharpness in W32.
92 0x5C	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W34 and W33**
	7:0		Average sharpness in W33.
	15:8		Average sharpness in W34.
93 0x5D	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W42 and W41.**
	7:0		Average sharpness in W41.
	15:8		Average sharpness in W42.
94 0x5E	**15:0**	**R/O**	**AF Filter 2 Average Sharpness Measure for AF Windows W44 and W43.**
	7:0		Average sharpness in W43.
	15:8		Average sharpness in W44.

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
128 0x80	**10:0**	**0x0160**	**Lens Correction Control**
	0	0	Sign for the K*F(x)*F(y). If 0, then K is positive; if 1, then K is negative.
	1	0	If 1, all the second X derivatives are doubled.
	2	0	If 1, all the second Y derivatives are doubled.
	3:5	100	Divisor for the first derivative, X direction. 000-/1, 001-/2, 010-/4, ... 111-/128. Also applies to second derivative.
	6:8	101	Divisor for the first derivative, Y direction. 000-/1, 001-/2, 010-/4, ... 111-/128. Also applies to second derivative.
	9	0	Shift column, LC specific.
	10	0	Shift row, LC specific.
	This register controls behavior of lens correction.		
129 0x81	**15:0**	**0x6432**	**Zone Boundaries X1 and X2**
	7:0		X2 boundary (/4) position.
	15:8		X1 boundary (/4) position.
	Definition of X1 and X2 boundaries		
130 0x82	**15:0**	**0x3296**	**Zone Boundaries X0 and X3**
	7:0		X0 boundary (/4) position.
	15:8		X3 boundary (/4) position.
	Definition of X0 and X3 boundaries.		
131 0x83	**15:0**	**0x9664**	**Zone Boundaries X4 and X5**
	7:0		X4 boundary (/4) position.
	15:8		X5 boundary (/4) position.
	Definition of X4 and X5 boundaries.		
132 0x84	**15:0**	**0x5028**	**Zone Boundaries Y1 and Y2**
	7:0		Y2 boundary (/4) position.
	15:8		Y1 boundary (/4) position.
	Definition of Y1 and Y2 boundaries.		
133 0x85	**15:0**	**0x2878**	**Zone Boundaries Y0 and Y3**
	7:0		Y0 boundary (/4) position.
	15:8		Y3 boundary (/4) position.
	Definition of Y0 and Y3 boundaries.		
134 0x86	**15:0**	**0x7850**	**Zone Boundaries Y4 and Y5**
	7:0		Y4 boundary (/4) position.
	15:8		Y5 boundary (/4) position.
	Definition of Y4 and Y5 boundaries.		
135 0x87	**15:0**	**0x0000**	**Center Offset**
	7:0		Offset of LC center in respect to geometrical center. X coordinate(/4).
	15:8		Offset of LC center in respect to geometrical center. Ycoordinate(/4).
	Offset of the central point for the lens correction (relative to the center of imaging array).		
136 0x88	**11:0**	**0x015E**	**F(x) for Red Color at the First Pixel of the Array**
137 0x89	**11:0**	**0x0143**	**F(x) for Green Color at the First Pixel of the Array**
138 0x8A	**11:0**	**0x0127**	**F(x) for Blue Color at the First Pixel of the Array**

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
139 0x8B	**11:0**	**0x012C**	**F(y) for Red Color at the First Pixel of the Array**
140 0x8C	**11:0**	**0x0103**	**F(y) for Green Color at the First Pixel of the Array**
141 0x8D	**11:0**	**0x00FD**	**F(y) for Blue Color at the First Pixel of the Array**
142 0x8E	**11:0**	**0x0CEF**	**dF/dx for Red Color at the First Pixel of the Array**
143 0x8F	**11:0**	**0x0D38**	**dF/dx for Green Color at the First Pixel of the Array**
144 0x90	**11:0**	**0x0D92**	**dF/dx for Blue Color at the First Pixel of the Array**
145 0x91	**11:0**	**0x0C18**	**dF/dy for Red Color at the First Pixel of the Array**
146 0x92	**11:0**	**0x0CF1**	**dF/dy for Green Color at the First Pixel of the Array**
147 0x93	**11:0**	**0x0D05**	**dF/dy for Blue Color at the First Pixel of the Array**
148 0x94	**15:0**	**0x0B03**	**Second Derivative for Zone 0 Red Color**
	7:0		d2F/dx2 for zone 0 red color.
	15:8		d2F/dy2 for zone 0 red color.
	Second derivative for red color in zone 0.		
149 0x95	**15:0**	**0x0000**	**Second Derivative for Zone 0 Green Color**
	7:0		d2F/dx2 for zone 0 green color.
	15:8		d2F/dy2 for zone 0 green color.
	Second derivative for green color in zone 0.		
150 0x96	**15:0**	**0x0100**	**Second Derivative for Zone 0 Blue Color**
	7:0		d2F/dx2 for zone 0 blue color.
	15:8		d2F/dy2 for zone 0 blue color.
	Second derivative for green color in zone 0.		
151 0x97	**15:0**	**0x2534**	**Second Derivative for Zone 1 Red Color**
	7:0		d2F/dx2 for zone 1 red color.
	15:8		d2F/dy2 for zone 1 red color.
	Second derivative for red color in zone 1.		
152 0x98	**15:0**	**0x1C33**	**Second Derivative for Zone 1 Green Color**
	7:0		d2F/dx2 for zone 1 green color.
	15:8		d2F/dy2 for zone 1 green color.
	Second derivative for green color in zone 1.		
153 0x99	**15:0**	**0x1A2E**	**Second Derivative for Zone 1 Blue Color**
	7:0		d2F/dx2 for zone 1 blue color.
	15:8		d2F/dy2 for zone 1 blue color.
	Second derivative for green color in zone 1.		
154 0x9A	**15:0**	**0x2A3A**	**Second Derivative for Zone 2 Red color**
	7:0		d2F/dx2 for zone 2 red color.
	15:8		d2F/dy2 for zone 2 red color.
	Second derivative for red color in zone 2.		

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
155 0x9B	15:0	0x252D	**Second Derivative for Zone 2 Green Color**
	7:0		d2F/dx2 for zone 2 green color.
	15:8		d2F/dy2 for zone 2 green color.
	Second derivative for green color in zone 2.		
156 0x9C	15:0	0x2823	**Second Derivative for Zone 2 Blue Color**
	7:0		d2F/dx2 for zone 2 blue color.
	15:8		d2F/dy2 for zone 2 blue color.
	Second derivative for green color in zone 2.		
157 0x9D	15:0	0x0F14	**Second Derivative for Zone 3 Red Color**
	7:0		d2F/dx2 for zone 3 red color.
	15:8		d2F/dy2 for zone 3 red color.
	Second derivative for red color in zone 3.		
158 0x9E	15:0	0x0D20	**Second Derivative for Zone 3 Green Color**
	7:0		d2F/dx2 for zone 3 green color.
	15:8		d2F/dy2 for zone 3 green color.
	Second derivative for green color in zone 3.		
159 0x9F	15:0	0x0421	**Second Derivative for Zone 3 Blue Color**
	7:0		d2F/dx2 for zone 3 blue color.
	15:8		d2F/dy2 for zone 3 blue color.
	Second derivative for green color in zone 3.		
160 0xA0	15:0	0x0D2A	**Second Derivative for Zone 4 Red Color**
	7:0		d2F/dx2 for zone 4 red color.
	15:8		d2F/dy2 for zone 4 red color.
	Second derivative for red color in zone 4.		
161 0xA1	15:0	0x1017	**Second Derivative for Zone 4 Green Color**
	7:0		d2F/dx2 for zone 4 green color.
	15:8		d2F/dy2 for zone 4 green color.
	Second derivative for green color in zone 4.		
162 0xA2	15:0	0x1617	**Second Derivative for Zone 4 Blue Color**
	7:0		d2F/dx2 for zone 4 blue color.
	15:8		d2F/dy2 for zone 4 blue color.
	Second derivative for green color in zone 4.		
163 0xA3	15:0	0x1642	**Second Derivative for Zone 5 Red Color**
	7:0		d2F/dx2 for zone 5 red color.
	15:8		d2F/dy2 for zone 5 red color.
	Second derivative for red color in zone 5.		
164 0xA4	15:0	0x1448	**Second Derivative for Zone 5 Green Color**
	7:0		d2F/dx2 for zone 5 green color.
	15:8		d2F/dy2 for zone 5 green color.
	Second derivative for green color in zone 5.		
165 0xA5	15:0	0x0F4E	**Second Derivative for Zone 5 Blue Color**
	7:0		d2F/dx2 for zone 5 blue color.
	15:8		d2F/dy2 for zone 5 blue color.
	Second derivative for green color in zone 5.		

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
166 0xA6	15:0	0x1F27	**Second Derivative for Zone 6 Red Color**
	7:0		d2F/dx2 for zone 6 red color.
	15:8		d2F/dy2 for zone 6 red color.
	Second derivative for green color in zone 6.		
167 0xA7	15:0	0x1A12	**Second Derivative for Zone 6 Green Color**
	7:0		d2F/dx2 for zone 6 green color.
	15:8		d2F/dy2 for zone 6 green color.
	Second derivative for green color in zone 6.		
168 0xA8	15:0	0x1E16	**Second Derivative for Zone 6 Blue Color**
	7:0		d2F/dx2 for zone 6 blue color.
	15:8		d2F/dy2 for zone 6 blue color.
	Second derivative for blue color in zone 6.		
169 0xA9	15:0	0x16C7	**Second Derivative for Zone 7 Red Color**
	7:0		d2F/dx2 for zone 7 red color.
	15:8		d2F/dy2 for zone 7 red color.
	Second derivative for red color in zone 7.		
170 0xAA	15:0	0x31C5	**Second Derivative for Zone 7 Green Color**
	7:0		d2F/dx2 for zone 7 green color.
	15:8		d2F/dy2 for zone 7 green color.
	Second derivative for green color in zone 7.		
171 0xAB	15:0	0x2AB6	**Second Derivative for Zone 7 Blue Color**
	7:0		d2F/dx2 for zone 7 blue color.
	15:8		d2F/dy2 for zone 7 blue color.
	Second derivative for blue color in zone 7.		
172 0xAC	15:0	0x0000	**X2 Factors**
	7:0		X2 factors for X zones. If 1 value of the value of d2F/dx2 is multiplied by 2.
	15:8		X2 factors for Y zones. If 1 value of the value of d2F/dy2 is multiplied by 2.
173 0xAD	7:0	0x0002	**Global Offset of F(x,y) Function** Signed 8 bit number. Value of 32 corresponds to gain of 1. Works as digital gain
174 0xAE	15:0	0x018E	**K Factor in K F(x) F(y)** This factor can increase lens compensation in the corners to up to 2 times.
192 0xC0	15:0	0	**Boundaries of First AE Measurement Window (Top/Left)**
	7:0		Left window boundary.
	15:8		Top window boundary.
	This register specifies top and left boundaries of the first from 16 (left-top) window used by AE measurement engine. The values programmed in the registers are desired boundaries divided by 8.		
193 0xC1	15:0	0x3C50	**Boundaries of First AE Measurement Window (Height/Width)**
	7:0		Window width.
	15:8		Window height.
	This register specifies height and width of the first from 16 window used by AE measurement engine. The values programmed in the registers are desired boundaries divided by 2.		
194 0xC2	15:0	0x4B00	**AE Measurement Window Size (LSW)**
	This register number of pixels in the window used by AE measurement engine.		

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
195 0xC3	**2:0**	**6**	**AE/AF Measurement Enable**
	0	0	MS bit of the AE measurement window size.
	1	1	AE statistic enable.
	2	1	AF statistic enable.
	This register specifies number of pixels in the window used by AE measurement engine.		
196 0xC4	**15:0**	**R/O**	**Average Luminance in AE Windows W12 and W11**
	7:0		Average Y in W11 (top left).
	15:8		Average Y in W12.
197 0xC5	**15:0**	**R/O**	**Average Luminance in AE Windows W14 and W13**
	7:0		Average Y in W13.
	15:8		Average Y in W14 (top right).
198 0xC6	**15:0**	**R/O**	**Average Luminance in AE Windows W22 and W21**
	7:0		Average Y in W21.
	15:8		Average Y in W21.
199 0xC7	**15:0**	**R/O**	**Average Luminance in AE Windows W24 and W23**
	7:0		Average Y in W23.
	15:8		Average Y in W24.
200 0xC8	**15:0**	**R/O**	**Average Luminance in AE Windows W32 and W31**
	7:0		Average Y in W31.
	15:8		Average Y in W32.
201 0xC9	**15:0**	**R/O**	**Average Luminance in AE Windows W34 and W33**
	7:0		Average Y in W33.
	15:8		Average Y in W34.
202 0xCA	**15:0**	**R/O**	**Average Luminance in AE Windows W42 and W41**
	7:0		Average Y in W41 (bottom left).
	15:8		Average Y in W43.
203 0xCB	**15:0**	**R/O**	**Average Luminance in AE Windows W44 and W43**
	7:0		Average Y in W43.
	15:8		Average Y in W44 (bottom right).
210 0xD2	**9:0**	**0x0194**	**Saturation and Color Kill**
	2:0	4	Saturation; 100 = 100%. Scales linearly with value.
	5:3	2	Color kill gain. Determines rate of gain decrease above threshold. If pixel value is above threshold the difference is multiplied by 2^ value-1 (for 0 factor is “0”) and subtracted from the base gain.
	8:6	6	Color kill threshold. Color kill effects only pixels with value larger then 128*value.
	9	0	Reserved.
211 0xD3	**15:0**	**0**	**Histogram Window Lower Boundaries**
	7:0		X0/8.
	15:8		Y0/8.
212 0xD4	**15:0**	**0x95C7**	**Histogram Window Upper Boundaries**
	7:0		X1/8.
	15:8		Y1/8.
213 0xD5	**10:0**	**0x030A**	**First Set of Bin Definitions**
	7:0		Offset for bin 0, divided by 4 on 10-bit scale.
	10:8		Bin width, 0-4LSB,1-8LSB,2-16LSB,…7-512LSB on a 10-bit scale.

Table 7: IFP Registers, Page 2 (continued)

Reg #	Bits	Default	Name
214 0xD6	**10:0**	**0x030A**	**Second Set of Bin Definitions**
	7:0		Offset for bin 0, divided by 4 on 10-bit scale.
	10:8		Bin width, 0-4LSB,1-8LSB,2-16LSB,…7-512LSB on a 10-bit scale.
215 0xD7	**13:0**	**0x000A**	**Histogram Window Size**
	12:0	1	Number of pixels in the window used by the histogram, set to width*height.
	13	0	Select a bin set to read. 0—bin set 1 1—bin set 2
216 0xD8	**15:0**	**0**	**Pixel Counts for Bin 0 and Bin 1**
	7:0		Pixel count for bin 0 (lowest values) divided window size, R215:2.
	15:8		Pixel count for bin 1 divided by window size, R215:2.
	Histogram module uses luma after interpolation. No color correction or gamma is applied. Digital gains and first black level are applied. This register may not be read out when image portion containing histogram window is being output. There are two bin sets which could be read through registers 216 and 217, based on value of bit 13 in reg 215[2].		
217 0xD9	**15:0**	**0x0002**	**Pixel Counts for Bin 2 and Bin 3**
	7:0		Pixel count for bin 2 divided by G factor.
	15:8		Pixel count for bin 3 (highest values) divided by G factor.
240 0xF0	**2:0**	**2**	**Page Register** 0 = sensor core 1 = IFP page 1 2 = IFP page 2
241 0xF1	**15:0**	**0**	**Bytewise Address** Special address to perform 16-bit reads and writes to the sensor in 8-bit chunks. See "8-Bit Write Sequence" on page 183.

JPEG Indirect Registers

Table 8: JPEG Indirect Registers (See Registers 30 and 31, Page 2)

Reg #	Bits	Default	Name
1 0x01	**2:0**	**0**	**JPEG Core Register 1**
	1:0	0	NCol: Number of color components—1.
	2	0	Re: When set, enables restart marker insertion into JPEG byte stream.
3 0x03	**15:0**	**0**	**JPEG Core Register 3 – NRST**
	Re-start interval - 1. Valid only when Re in Core Register 1 is set. This register defines the number of MCUs between Restart Markers.		
4 0x04	**7:0**	**0**	**JPEG Core Register 4**
	0	0	HD0: DC Huffman table pointer for color component 0.
	1	0	HA0: AC Huffman table pointer for color component 0.
	3:2	0	QT0: Quantization table pointer for color component 0.
	7:4	0	NBlock0: Number of 8 x 8 blocks—1 for color component 0 in MCU.
5 0x05	**7:0**	**0**	**JPEG Core Register 5**
	0	0	HD1: DC Huffman table pointer for color component 1.
	1	0	HA1: AC Huffman table pointer for color component 1.
	3:2	0	QT1: Quantization table pointer for color component 1.
	7:4	0	NBlock1: Number of 8 x 8 blocks—1 for color component 1 in MCU.
6 0x06	**7:0**	**0**	**JPEG Core Register 6**
	0	0	HD2: DC Huffman table pointer for color component 2.
	1	0	HA2: AC Huffman table pointer for color component 2.
	3:2	0	QT2: Quantization table pointer for color component 2.
	7:4	0	NBlock2: Number of 8 x 8 blocks—1 for color component 2 in MCU.
7 0x07	**7:0**	**0**	**JPEG Core Register 7**
	0	0	HD3: DC Huffman table pointer for color component 3.
	1	0	HA3: AC Huffman table pointer for color component 3.
	3:2	0	QT3: Quantization table pointer for color component 3.
	7:4	0	NBlock3: Number of 8 x 8 blocks—1 for color component 3 in MCU.
8 0x08	**15:0**	**0**	**JPEG Core Register8 – NMCU LSB**
	Low-order word of NMCU (NMCU = number of MCUs contained in the image to be encoded minus 1).		
9 0x09	**9:0**	**0**	**JPEG Core Register9 – NMCU_MSB**
	High-order word of NMCU (NMCU = number of MCUs contained in the image to be encoded minus 1).		
128-511 0x080 - 0x1FF	**13:0**	**0**	**JPEG Quantization Memory**
	0 - 63—Quantization table 0 64 - 127—Quantization table 1 128 - 191—Quantization table 2 192 - 255—Quantization table 3 256 - 319—Quantization table 4 320 - 383—Quantization table 5		
512 - 895 0x200 - 0x37F	**11:0**	**0**	**JPEG Huffman Memory**
	0 - 175—AC Huffman table 0 176 - 351—AC Huffman table 1 352 - 367—DC Huffman table 0 368 - 383—DC Huffman table 1		
1024 - 1407 0x400 - 0x43F	**13:0**	**0**	**JPEG DCT Memory**
	64 x 14 dual-port RAM is accessible to host and microcontroller indirectly for testing purposes.		

Table 8: JPEG Indirect Registers (See Registers 30 and 31, Page 2) (continued)

Reg #	Bits	Default	Name
1408 - 1471 0x440 - 0x47F	13:0	0	JPEG Zigzag Memory 0
	64 x 14 dual-port RAM is accessible to host and microcontroller indirectly for testing purposes.		
1472 - 1535 0x480 - 0x4BF	13:0	0	JPEG Zigzag Memory 1
	64 x 14 dual-port RAM is accessible to host and microcontroller indirectly for testing purposes.		

Firmware Driver Variables

Table 9: Drivers IDs

ID	VARNAME	Description
0	mon	Monitor
1	seq	Sequencer
2	ae	Auto exposure
3	awb	Auto white balance
4	fd	Flicker detection
5	af	Auto focus
6	afm	Auto focus mechanics
7	mode	Mode
8		Reserved
9	jpeg	JPEG
11	hg	Histogram
12–15		Reserved
Driver Extensions		
16		Monitor extension
17		Sequencer extension
18		Auto exposure extension
19		Auto white balance extension
20		Flicker detection extension
21		Auto focus extension
22		Auto focus mechanics extension
23		Mode extension
24		Reserved
25		JPEG extension
26		Histogram extension
27–31		Reserved

Monitor

Table 10: Driver Variables–Monitor Driver (ID = 0)

Offs	Name	Type	Default	RW	Description
0	vmt	void*	58224	RW	Reserved
2	cmd	char	0	RW	Monitor command Setting this variable to 1 causes the monitor to call a firmware function whose address equals mon.arg1. If the function needs an argument, it is given the value of mon.arg2. If the function returns a 1- or 2-byte value, it is put in mon.arg2. Setting mon.cmd to 2 causes the monitor to calculate CRC for a memory block defined by a starting address in mon.arg1 and length in mon.arg2. The CRC is returned in mon.arg2. The monitor starts executing each received command at the earliest opportunity. Before starting, it sets bit 7 of mon.cmd to 1. The bit remains set until the command execution is finished. Usage: Set mon.arg1 and mon.arg2 and write command number to mon.cmd. Wait until mon.cmd=0. Retrieve numerical result, if any, from mon.arg2.
3	arg1	uint	0	RW	First argument for monitor command
5	arg2	uint	0	RW	Second argument for monitor command
7	msgCount	char	0	R	Number of posted messages Usage: Poll this variable to detect messages. When mon.msgCount > 0, read mon.msg. TBD—clear counter.
8	msg	long	0	R	First message (unused)
12	ver	uchar	32	R	Bits [6:0] = firmware version Bit 7 = reserved

Sequencer

Table 11: Driver Variables–Sequencer Driver (ID = 1)

Offs	Name	Type	Default	RW	Description
0	vmt	void*	61547	RW	Reserved
2	mode	uchar	0x0F	RW	Set of 1-bit switches enabling various drivers and outdoor white balance option. Writing 1 to a particular switch enables the corresponding driver or option. Bit 0—Auto exposure driver (ID = 2) Bit 1—Flicker detection driver (ID = 4) Bit 2—Auto white balance driver (ID = 3) Bit 3—Histogram driver (ID = 11) Bit 4—Auto focus driver (ID = 5) Bit 5—Option to force outdoor white balance settings if exposure value exceeds sharedParams.outdoorTH threshold. Bits 6:7—New AF scheme that improves performance with noise dominated windows. Bit 7—TBD
3	cmd	uchar	0	RW	Command or program to execute 0—Run 1—Do Preview 2—Do Capture 3—Do Standby 4—Do lock 5—Refresh 6—Refresh mode Writing a positive value to this variable commands the sequencer to execute the corresponding program. The sequencer resets cmd to 0, executes the program, and then resumes running in its current state.
4	state	uchar	3	R	Current state of sequencer 0—Initialize 1—Mode Change to Preview 2—Enter Preview 3—Preview 4—Leave Preview 5—Mode Change to Capture 6—Enter Capture 7—Capture 8—Leave Capture 9—Standby
5	stepMode	uchar	0	RW	Step-by-step mode for sequencer Bit 0—Step mode On/Off (1 = On) Bit 1—1 forces the sequencer to do next step
6	sharedParams.flashType	uchar	0	RW	Type of flash to be used 0—None 1—LED 2—Xenon 3—Xenon burst Bit 7—1if flash was on in LOCK mode

Table 11: Driver Variables–Sequencer Driver (ID = 1) (continued)

Offs	Name	Type	Default	RW	Description
7	sharedParams.aeContBuff	uchar	8	RW	Weighting factor determining to what extent AE driver averages luma over time in continuous AE mode. Setting of 32 disables luma averaging—only current luma values are used. Lower settings enable luma averaging.
8	sharedParams.aeContStep	uchar	2	RW	Number of steps in which AE driver is expected to reach target brightness in continuous AE mode.
9	sharedParams.aeFastBuff	uchar	32	RW	Weighting factor determining to what extent AE driver averages luma over time in fast mode. Setting of 32 disables luma averaging—only current luma values are used. Lower settings enable luma averaging.
10	sharedParams.aeFastStep	uchar	1	RW	Number of steps in which AE driver is expected to reach target brightness in fast AE mode.
11	sharedParams.awbContBuff	uchar	8	RW	Weighting factor determining to what extent AWB driver averages luma over time in continuous AWB mode. Setting of 32 disables luma averaging—only current luma values are used. Lower settings enable luma averaging.
12	sharedParams.awbContStep	uchar	2	RW	Number of steps in which AWB driver is expected to reach target color balance in continuous AWB mode.
13	sharedParams.awbFastBuff	uchar	32	RW	Weighting factor determining to what extent AWB driver averages luma over time in fast mode. Setting of 32 disables luma averaging—only current luma values are used. Lower settings enable luma averaging.
14	sharedParams.awbFastStep	uchar	1	RW	Number of steps in which AWB driver is expected to reach target color balance in fast AWB mode.
18	sharedParams.totalMaxFrames	uchar	30	RW	Number of frames after which every driver must time out.
19	sharedParams.flashTH	uchar	0	RW	Exposure value (EV) threshold. If current EV is below this threshold, flash is used in autoevaluate flash mode.
20	sharedParams.outdoorTH	uchar	10	RW	Exposure value (EV) threshold. If current EV is above this threshold and bit 5 of seq.mode equals 1, AWB driver is forced to choose color correction settings suitable for daylight color temperature of 6500 K.
21	sharedParams.LLmode	uchar	0x60	RW	Bit 0—change interpolation threshold Bit 1—reduce color saturation Bit 2—reduce aperture correction Bit 3—increase aperture correction threshold Bit 4—enable Y filter
22	sharedParams.LLvirtGain1	uchar	0x51	RW	Min. LL virtual gain. Defined as (ae.VirtGain/2 + ae.DGainAE1/16 + ae.DGainAE2/4).
23	sharedParams.LLvirtGain2	uchar	0x5A	RW	Max. LL virtual gain. Min. and max. LL virtual gains define when low-light parameters start and stop changing.
24	sharedParams.LLSat1	uchar	128	RW	Max. color saturation for color correction matrix (128 means 100%).
25	sharedParams.LLSat2	uchar	0	RW	Min. color saturation
26	sharedParams.LLInterpThresh1	uchar	16	RW	Min. threshold for interpolation
27	sharedParams.LLInterpThresh2	uchar	64	RW	Max. threshold for interpolation

Table 11: Driver Variables–Sequencer Driver (ID = 1) (continued)

Offs	Name	Type	Default	RW	Description
28	sharedParams.LLApCorr1	uchar	2	RW	Max. aperture correction
29	sharedParams.LLApCorr2	uchar	0	RW	Min. aperture correction
30	sharedParams.LLApThresh1	uchar	8	RW	Min. aperture correction threshold
31	sharedParams.LLApThresh2	uchar	64	RW	Max. aperture correction threshold
32	captureParams.mode	uchar	0x00	RW	Capture mode Bit 0—capture with Xenon flash (still only) Bit 1—capture video Bit 2— reserved Bit 3—AF On (video only) Bit 4—AE On (video only) Bit 5—AWB On (video only) Bit 6—HG On (video only)
33	captureParams.numFrames	uchar	3	RW	Number of frames captured in still capture mode
34	previewParams[0].ae	uchar	1	RW	Auto exposure configuration in PreviewEnter state 0—Off 1—Fast settling 2—Manual 3—Continuous 4—Fast settling + metering
35	previewParams[0].fd	uchar	0	RW	Flicker detection configuration in PreviewEnter state 0—Off 1—Manual 2—Continuous
36	previewParams[0].awb	uchar	1	RW	Auto white balance configuration in PreviewEnter state 0—Off 1—Fast settling 2—Manual 3—Continuous
37	previewParams[0].af	uchar	0	RW	Auto focus configuration in PrevieweEnter state 0—Off 1—Fast 2—Manual 5—Creep compensation
38	previewParams[0].hg	uchar	1	RW	Histogram configuration in PreviewEnter state 0—Off 1—Fast 2—Manual 3—Continuous
39	previewParams[0].flash	uchar	0	RW	Flash configuration in PreviewEnter state 0 —Off 1—On 2—Locked 3—AutoEvaluate Bit 7—Load user defined settings
40	previewParams[0].skipframe	uchar	0x00	RW	Bit 7—1means turn off fen Bit 6—1means skip state Bit 5—1means skip first frame when LED On

Table 11: Driver Variables–Sequencer Driver (ID = 1) (continued)

Offs	Name	Type	Default	RW	Description
41	previewParams[1].ae	uchar	3	RW	Auto exposure configuration in Preview state 0—Off 1—Fast settling 2—Manual 3—Continuous 4—Fast settling + metering
42	previewParams[1].fd	uchar	2	RW	Flicker detection configuration in Preview state 0—Off 1—Manual 2—Continuous
43	previewParams[1].awb	uchar	3	RW	Auto white balance configuration in Preview state 0—Off 1—Fast settling 2—Manual 3—Continuous
44	previewParams[1].af	uchar	0	RW	Auto focus configuration in Preview state 0—Off 1—Fast 2—Manual 5—Creep compensation
45	previewParams[1].hg	uchar	3	RW	Histogram configuration in Preview state 0—Off 1—Fast 2—Manual 3—Continuous
46	previewParams[1].flash	uchar	0	RW	Flash configuration in Preview state 0—Off 1—On 2—Locked 3—AutoEvaluate Bit 7—Load user defined settings
47	previewParams[1].skipframe	uchar	0x00	RW	Bit 7—1means turn off fen Bit 6—1means skip state Bit 5—1means skip first frame when LED On
48	previewParams[2].ae	uchar	4	RW	Auto exposure configuration in PreviewLeave state 0—Off 1—Fast settling 2—Manual 3—Continuous 4—Fast settling + metering
49	previewParams[2].fd	uchar	0	RW	Flicker detection configuration in PreviewLeave state 0—Off 1—Manual 2—Continuous
50	previewParams[2].awb	uchar	1	RW	Auto white balance configuration in PreviewLeave state 0—Off 1—Fast settling 2—Manual 3—Continuous

Table 11: Driver Variables–Sequencer Driver (ID = 1) (continued)

Offs	Name	Type	Default	RW	Description
51	previewParams[2].af	uchar	0	RW	Auto focus configuration in PreviewLeave state 0—Off 1—Fast 2—Manual 5—Creep compensation
52	previewParams[2].hg	uchar	1	RW	Histogram configuration in PreviewLeave state 0—Off 1—Fast 2 —Manual 3—Continuous
53	previewParams[2].flash	uchar	0	RW	Flash configuration in PreviewLeave state 0—Off 1—On 2—Locked 3—AutoEvaluate Bit 7—Load user defined settings.
54	previewParams[2].skipframe	uchar	0x00	RW	Bit 7—1means turn off fen Bit 6—1means skip state Bit 5—1means skip first frame when LED On
55	previewParams[3].ae	uchar	0	RW	Auto exposure configuration in CaptureEnter state 0—Off 1—Fast settling 2—Manual 3—Continuous 4—Fast settling + metering
56	previewParams[3].fd	uchar	0	RW	Flicker detection configuration in CaptureEnter state 0—Off 1—Manual 2—Continuous
57	previewParams[3].awb	uchar	0	RW	Auto white balance configuration in CaptureEnter state 0—Off 1—Fast settling 2—Manual 3—Continuous
58	previewParams[3].af	uchar	0	RW	Auto focus configuration in CaptureEnter state 0—Off 1—Fast 2—Manual 5—Creep compensation
59	previewParams[3].hg	uchar	0	RW	Histogram configuration in CaptureEnter state 0—Off 1—Fast 2—Manual 3—Continuous
60	previewParams[3].flash	uchar	0	RW	Flash configuration in CaptureEnter state 0—Off 1—On 2—Locked 3—AutoEvaluate Bit 7—Load user defined settings

Table 11: Driver Variables–Sequencer Driver (ID = 1) (continued)

Offs	Name	Type	Default	RW	Description
61	previewParams[3].skipframe	uchar	0x00	RW	Bit 7—1means turn off fen Bit 6—1means skip state Bit 5—1means skip first frame when LED On

Auto Exposure

Table 12: Driver Variables–Auto Exposure Driver (ID = 2)

Offs	Name	Type	Default	RW	Description
0	vmt	void*	59518	RW	Reserved
2	windowPos	uchar	0	RW	Position of upper left corner of first AE window Bits [3:0]—X0 in units of 1/16th of frame width Bits [7:4]—Y0 in units 1/16th of frame height New position written to this variable takes effect only after REFRESH_MODE command is given to the sequencer (seq.cmd is set to 6).
3	windowSize	uchar	0xEF	RW	Dimensions of 4 x 4 array of AE windows Bits [3:0]—width (in units of 1/16th of frame width) decremented by 1 Bits [7:4]—height (in units of 1/16th of frame height) decremented by 1 New dimensions written to this variable take effect only after REFRESH_MODE command is given to the sequencer (seq.cmd is set to 6).
4	wakeUpLine	uint		R	Number of image row at which MCU wakes up to do AE calculations. This number is automatically adjusted after each change of the position and dimensions of the AE windows.
6	Target	uchar	60	RW	Target brightness
7	Gate	uchar	10	RW	AE sensitivity
8	SkipFrames	uchar	0	RW	Frequency of AE updates
9	JumpDivisor	uchar	2	RW	Factor reducing exposure jumps (1 = fastest jumps)
10	lumaBufferSpeed	uchar	8	RW	Speed of buffering (32 = fastest, 1= slowest)
11	maxR12	uint	1200	RW	Maximum value of R12:0 (shutter delay)
13	minIndex	uchar	0	RW	Minimum allowed zone number (i.e. minimum integration time)
14	maxIndex	uchar	24	RW	Maximum allowed zone number (i.e. maximum integration time)
15	minVirtGain	uchar	16	RW	Minimum allowed virtual gain
16	maxVirtGain	uchar	128	RW	Maximum allowed virtual gain
17	maxADChi	uchar	13	RW	Maximum ADC Vref_hi setting
18	minADChi	uchar	11	RW	Minimum ADC Vref_hi setting
19	minADClo	uchar	7	RW	Minimum ADC Vref_lo setting
20	maxDGainAE1	uint	128	RW	Maximum digital gain pre-LC
22	maxDGainAE2	uchar	32	RW	Maximum digital gain post-LC
23	IndexTH23	uchar	8	RW	Zone number to start gain increase in low-light. Sets frame rate at normal illumination.
24	maxGain23	uchar	120	RW	Maximum gain to increase in low-light before dropping frame rate. Sets frame rate at low-light.

Table 12: Driver Variables–Auto Exposure Driver (ID = 2) (continued)

Offs	Name	Type	Default	RW	Description
25	weights	uchar	0xFF	RW	Bits [7:4]—background weight Bits [3:0]—center zone weight
26	status	uchar	192	RW	AE status Bit 0—1 if AE reached the limit Bit 1—1 if R9 is changed (need to skip frame) Bit 2—ready Bit 3—do mettering mode Bit 4—mettering mode is done Bit 6—on/Off overexpose fit Bit 7—on/Off overexpose compensation
27	CurrentY	uchar		R	Last measured luminance
28	R12	uint		RW	Current shutter delay value
30	Index	uchar		R	Current zone (integration time)
31	VirtGain	uchar		RW	Current virtual gain
32	ADC_hi	uchar		R	Current ADC VREF_HI value
33	ADC_lo	uchar		R	Current ADC VREF_LO value
34	DGainAE1	uint		RW	Current digital gain pre-LC
36	DGainAE2	uchar		RW	Current digital gain post-LC
37	R9	uint		RW	Current R9:0 value (integration time)
39	R65	uchar		RW	Current ADC VREF values
40	rowTime	uint	531	RW	Row time divided by 4 (in master clock periods)
42	gainR12	uchar	128	R	Gain compensating too low maxR12 (128 = unit gain)
43	SkipFrames_cnt	uchar		R	Counter of skipped frames
44	BufferedLuma	uint		R	Current time-buffered measured luma
46	dirAE_prev	uchar		R	Previous state of AE
47	R9_step	uint	157	RW	Integration time of one zone
50	maxADClo	uchar		R	Maximum ADC Vref_lo setting
51	physGainR	uint		R	Physical (analog) R gain
53	physGainG	uint		R	Physical (analog) G gain
55	physGainB	uint		R	Physical (analog) B gain
82	mmEVZone1	uchar	16	RW	EV threshold for scenes containing the sun
83	mmEVZone2	uchar	12	RW	EV threshold for bright outdoor scenes
84	mmEVZone3	uchar	8	RW	EV threshold for indoor or outdoor twilight scenes
85	mmEVZone4	uchar	2	RW	EV threshold for night scenes
86	mmShiftEV	uchar	13	RW	Exposure Value shift. Adjust this value for given lens.
87	numOE	uchar		R	Number of overexposed zones

Auto White Balance

Table 13: Driver Variables–Auto White Balance (ID = 3)

Offs	Name	Type	Default	RW	Description
0	vmt	void*	59731	RW	Reserved
2	windowPos	uchar	0	RW	Position of upper left corner of AWB window Bits [3:0]—X0 in units of 1/16th of frame width Bits [7:4]—Y0 in units 1/16th of frame height New position written to this variable takes effect only after REFRESH_MODE command is given to the sequencer (seq.cmd is set to 6).
3	windowSize	uchar	0xEF	RW	Dimensions of AWB window Bits [3:0]—width (in units of 1/16th of frame width) decremented by 1. Bits [7:4]—height (in units of 1/16th of frame height) decremented by 1. New dimensions written to this variable take effect only after REFRESH_MODE command is given to the sequencer (seq.cmd is set to 6).
4	wakeUpLine	uint		R	Number of image row at which MCU wakes up to perform AWB calculations. This number is automatically adjusted after each change of the position and dimensions of the AWB window.
6	ccmL[0]	int	0x0219	RW	Left color correction matrix element K11. Value is encoded in signed two's complement form; 256 means 1.0. New values written to awb.ccmL array take effect only after REFRESH command is given to the sequencer (seq.cmd is set to 5).
8	ccmL[1]	int	0xFEEC	RW	Left color correction matrix element K12
10	ccmL[2]	int	0xFFFF	RW	Left color correction matrix element K13
12	ccmL[3]	int	0xFF6A	RW	Left color correction matrix element K21
14	ccmL[4]	int	0x01D4	RW	Left color correction matrix element K22
16	ccmL[5]	int	0xFFC9	RW	Left color correction matrix element K23
18	ccmL[6]	int	0xFF71	RW	Left color correction matrix element K31
20	ccmL[7]	int	0xFDD3	RW	Left color correction matrix element K32
22	ccmL[8]	int	0x03ED	RW	Left color correction matrix element K33
24	ccmL[9]	int	0x0020	RW	Left color correction matrix red/green gain ratio. Value is interpreted as unsigned; 32 means 1.0.
26	ccmL[10]	int	0x0035	RW	Left color correction matrix blue/green gain ratio. Value is interpreted as unsigned; 32 means 1.0.
28	ccmRL[0]	int	0xFF92	RW	Delta color correction matrix element D11. Value is encoded in signed two's complement form; 256 means 1.0. New values written to awb.ccmRL array take effect only after REFRESH command is given to the sequencer (seq.cmd is set to 5).
30	ccmRL[1]	int	0x0098	RW	Delta color correction matrix element D12
32	ccmRL[2]	int	0xFFE1	RW	Delta color correction matrix element D13
34	ccmRL[3]	int	0x0014	RW	Delta color correction matrix element D21
36	ccmRL[4]	int	0xFFFC	RW	Delta color correction matrix element D22
38	ccmRL[5]	int	0xFFF2	RW	Delta color correction matrix element D23
40	ccmRL[6]	int	0x004E	RW	Delta color correction matrix element D31
42	ccmRL[7]	int	0x0148	RW	Delta color correction matrix element D32
44	ccmRL[8]	int	0xFE3F	RW	Delta color correction matrix element D33

Table 13: Driver Variables–Auto White Balance (ID = 3) (continued)

Offs	Name	Type	Default	RW	Description
46	ccmRL[9]	int	0x000A	RW	Delta color correction matrix red/green gain ratio. Value is interpreted as unsigned; 32 means 1.0.
48	ccmRL[10]	int	0xFFF1	RW	Delta color correction matrix blue/green gain ratio. Value is interpreted as unsigned; 32 means 1.0.
50	ccm[0]	int		RW	Current color correction matrix element C11. Value is stored in signed two's complement form; 256 means 1.0.
52	ccm[1]	int		RW	Current color correction matrix element C12
54	ccm[2]	int		RW	Current color correction matrix element C13
56	ccm[3]	int		RW	Current color correction matrix element C21
58	ccm[4]	int		RW	Current color correction matrix element C22
60	ccm[5]	int		RW	Current color correction matrix element C23
62	ccm[6]	int		RW	Current color correction matrix element C31
64	ccm[7]	int		RW	Current color correction matrix element C32
66	ccm[8]	int		RW	Current color correction matrix element C33
68	ccm[9]	int		RW	Current color correction matrix red/green gain ratio. Value is interpreted as unsigned; 32 means 1.0.
70	ccm[10]	int		RW	Current color correction matrix blue/green gain ratio. Value is interpreted as unsigned; 32 means 1.0.
72	GainBufferSpeed	uchar	8	RW	Speed of time-buffering (32 = fastest, 1= slowest)
	JumpDivisor	uchar	2	RW	Derating factor for white balance changes (1= fastest changes)
74	GainMin	uchar	83	RW	Minimum allowed AWB digital gain (128 means 1.0)
75	GainMax	uchar	192	RW	Maximum allowed AWB digital gain (128 means 1.0)
76	GainR	uchar		R	Current R digital gain (128 means1.0)
77	GainG	uchar		R	Current G digital gain (128 means1.0)
78	GainB	uchar		R	Current B digital gain (128 means 1.0)
79	CCMpositionMin	uchar	0	RW	Leftmost position of color correction matrix (corresponding to incandescent light)
80	CCMpositionMax	uchar	127	RW	Rightmost position of color correction matrix (corresponding to daylight)
81	CCMposition	uchar	64	RW	Current position of color correction matrix (0 corresponds to incandescent light, 127 to daylight)
82	saturation	uchar	128	RW	Saturation of color correction matrix (128 means 100%)
83	mode	uchar	0	RW	Bit 0—1 = AWB is in steady state Bit 1—1 = limits are reached Bit 2—1 = reserved Bit 3—1 = debug Bit 4—1 = no awb statistic Bit 5—1 = force AWB DGains to unit; program value into awb.CCMPosition to manually set matrix position Bit 6—1 = disable CCM normalization Bit 7—1 = force outdoor settings, set by the sequencer
84	GainR_buf	uint	0x1000	RW	Time-buffered R gain (0x1000 means 1.0)
86	GainB_buf	uint	0x1000	RW	Time-buffered B gain (0x1000 means 1.0)
88	sumR	uchar		R	AWB statistics (normalized to an arbitrary number)
89	sumY	uchar		R	AWB statistics (normalized to an arbitrary number)
90	sumB	uchar		R	AWB statistics (normalized to an arbitrary number)

Table 13: Driver Variables Auto White Balance (ID = 3) (continued)

Offs	Name	Type	Default	RW	Description
91	steadyBGainOutMin	uchar	115	RW	Blue gain min. threshold to start search. When the blue gain falls below this threshold, the AWB driver starts searching for new color correction matrix position. Threshold setting of 128 corresponds to gain of 1.0; higher setting means higher gain.
92	steadyBGainOutMax	uchar	140	RW	Blue gain max. threshold to start search. When the blue gain goes above this threshold, the AWB driver starts searching for new color correction matrix position. Threshold setting of 128 corresponds to gain of 1.0; higher setting means higher gain.
93	steadyBGainInMin	uchar	124	RW	Blue gain min. threshold to end search. When the blue gain is between awb.steadyBGainInMin and awb.steadyBGainInMax, the AWB driver maintains current color correction matrix position. Threshold setting of 128 corresponds to gain of 1.0; higher setting means higher gain.
94	steadyBGainInMax	uchar	131	RW	Blue gain max. threshold to end search. See awb.steadyBGainInMin above.
95	cntPxlTH	uint	100	RW	Reserved
97	TG_min0	uchar	226	RW	Reserved
98	TG_max0	uchar	246	RW	Reserved
99	XO		16	RW	CCM position threshold between Day and Incandescent normalization coefficients.
100	kR_L	uchar	170	RW	CCM red row normalization coefficient for left matrix position (128 - minimal number)
101	kG_L	uchar	150	RW	CCM green row normalization coefficient for left matrix position (128 - minimal number)
102	kB_L	uchar	128	RW	CCM blue row normalization coefficient for left matrix position (128 - minimal number)
103	kR_R	uchar	128	RW	CCM red row normalization coefficient for right matrix position (128 - minimal number)
104	kG_R	uchar	128	RW	CCM green row normalization coefficient for right matrix position (128 - minimal number)
105	kB_R	uchar	128	RW	CCM blue row normalization coefficient for right matrix position (128 - minimal number)

Flicker Detection

Table 14: Driver Variables–Flicker Detection Driver (ID = 4)

Offs	Name	Type	Default	RW	Description
0	vmt	void*	59884	RW	Reserved
2	windowPosH	uchar	29	RW	Width of flicker measurement window and position of its left boundary X0 Bits [3:0]—width (in units of 1/16th of frame width) decremented by 1 Bits [7:4]—X0 (in the same units as the width) At the beginning of each flicker measurement, the top boundary (Y0) of the flicker measurement window is set at row 64. Average luminance in the window is measured by the IFP measurement engine and stored in fd.Buffer[0]. Then Y0 is incremented by fd.windowHeight and the buffer index by 1. By repeating these steps 47 times, the entire fd.Buffer is filled with average luminance values from a vertical array of 48 equal-size windows.
3	windowHeight	uchar	4	RW	Bits [5:0]—flicker measurement window height in rows
4	mode	uchar	0	RW	Mode switches and indicators Bit 7 —1 enables manual mode, 0 disables it Bit 6 —0 selects 60Hz settings for manual mode, 1 selects 50Hz settings Bit 5 —read-only, indicates current settings (0–60Hz, 1–50Hz) Bit 4—1 enables debug mode, 0 disables it Bits [3:0]—read-only, reserved for auto FD
5	wakeUpLine	uint	64	R	Number of image row at which MCU wakes up to perform flicker detection. Changing the value of fd.wakeUpLine has no effect on the functioning of the FD driver, because it does not use this variable. It simply sets it to 64 at initialization, to indicate the top of the flicker measurement area. See fd.windowPosH above.
7	smooth_cnt	uchar	5	RW	Reserved
8	search_f1_50	uchar	30	RW	Lower limit of period range of interest in 50Hz flicker detection. If the FD driver is searching for 50Hz flicker and detects in a frame a flicker-like signal whose period in rows is between fl.search_f1_50 and fl.search_f2_50, it counts the frame as one containing flicker.
9	search_f2_50	uchar	32	RW	Upper limit of period range of interest in 50Hz flicker detection. See fd.search_f1_50 above.
10	search_f1_60	uchar	37	RW	Lower limit of period range of interest in 60Hz flicker detection. If the FD driver is searching for 60Hz flicker and detects in a frame a flicker-like signal whose period in rows is between fl.search_f1_60 and fl.search_f2_60, it counts the frame as one containing flicker.
11	search_f2_60	uchar	39	RW	Upper limit of period range of interest in 60Hz flicker detection. See fd.search_f1_60 above.
12	skipFrame	uchar	0	RW	Skip frame before subtracting two frames.
13	stat_min	uchar	3	RW	Flicker is considered detected if fd.stat_min out of fd.stat_max consecutive frames contain flicker (periodic signal of sought frequency).
14	stat_max	uchar	5	RW	See fl.stat_min.

Table 14: Driver Variables–Flicker Detection Driver (ID = 4) (continued)

Offs	Name	Type	Default	RW	Description
15	stat	uchar		R	Reserved
16	minAmplitude	uchar	5	RW	Reserved
17	R9_step60	uint	157	RW	Minimal shutter width step for 60Hz AC
19	R9_step50	uint	188	RW	Minimal shutter width step for 50Hz AC
21	Buffer[48]	uchar[48]		R	Reserved

Auto Focus

Table 15: Driver Variables-Auto Focus Driver (ID = 5)

Off	Name	Type	Default	RW	Description
0	Vmt	void*	0xE9D2	RW	Pointer to virtual method table (VMT)
2	windowPos	uchar	0x44	RW	Position of the upper left corner of the first AF window (W11): Bits [3:0]—x coordinate (horizontal offset from the upper left corner of the frame) in units of 1/16th of frame width, Bits [7:4]—y coordinate (vertical offset from the upper left corner of the frame) in units of 1/16th of frame height. New position written to this variable takes effect after REFRESH_MODE command is given to Sequencer driver (sq.cmd is set to 6).
3	windowSize	uchar	0x77	RW	Dimensions of the 4 x 4 array of AF windows: Bits [3:0]—width (in units of 1/16th of frame width) decremented by 1, Bits [7:4]—height (in units of 1/16th of frame height) decremented by 1. New dimensions written to this variable take effect only after REFRESH_MODE command is given to the Sequencer (sq.cmd is set to 6).
4	mode	uchar	0	RW	Two mode switches and 5 bits reserved for use in default snapshot AF mode: Bit 7—manual mode switch (0—manual mode disabled, 1—enabled), Bit 6—creep compensation mode switch (0 - creep compensation mode disabled, 1—enabled), Bits [4:0]—reserved for use in snapshot AF mode. If AF is enabled in the Sequencer (sq.mode bit 4 = 1) and manual mode is disabled (af.mode bit 7 = 0), a snapshot AF sequence can be triggered at any time by setting af.mode bit 0 to 1. Bits [4:0] are used in the sequence and automatically cleared at its end.
5	modeEx	uchar	128	RW	Four option switches and 4 status indicators: Bit 7—switch enabling the second flyback (jump to the start position of the first scan) and then jump to best focus position, Bit 6—switch enabling the second flyback and retracing of scan steps to best focus position (0—option disabled, 1—enabled), Bit 5— switch enabling the second scan (0—disabled, 1— enabled), Bit 4— error flag to indicate that AF has failed due to insufficient variability of sharpness scores with lens positions (see af.shaTH). Bit 3—if bit 7 of af.mode equals 0, this bit enables skipping 1 extra frame after detecting that bit 1 of afm.status has been cleared (i.e. after the end of every lens movement), Bit 2—status indicator, set to 1 when the extra frame is being skipped, Bit 1—status indicator, set to 1 when the second scan is in progress, Bit 0—status indicator, set to 1 when sharpness scores are ready.
6	numSteps	uchar	10	RW	Number of steps (lens positions tried) in the first scan.

Table 15: Driver Variables-Auto Focus Driver (ID = 5) (continued)

Off	Name	Type	Default	RW	Description
7	initPos	uchar	0	RW	Number (index) of start position, af.positions[af.initPos], used in the first scan and optional second scan. Must be 0 at the beginning of the first scan if the second is enabled. Otherwise, can be set before the first scan to any value below 20-af.numSteps. The AF driver makes af.initPos equal to af.numSteps at the beginning of the second scan and equal to 0 at the end of last scan (first or second, whichever is last).
8	numSteps2	uchar	6	RW	Bits [3:0]—desired number of steps in second scan (max. allowed number is 14) Bits [7:4]—actual number of steps in the second scan (calculated by the AF driver at the beginning of the scan).
9	stepSize	uchar	6	RW	Logical step size for the second scan. Because the logical range of motion is from 0 to 255, af.stepSize=6 means that during the second scan the AF driver tries to move the lens in increments equal to 6/255 of the length of its entire motion range. However, the lens actuator may or may not be able to move the lens in steps of precisely that size. It is important to ascertain that lens movements requested by the AF driver during the second scan can be at least reasonably approximated by the lens actuator. The quality of the approximation may depend on how well the physical limitations of the actuator are accounted for in the source code and /or configuration of the AFM driver.
10	wakeUpLine	uint	448	RW	Number of image row at which the MCU wakes up to execute AF driver code. When the function AF_SetSize resizes the 4 x 4 array of AF windows according to new values of af.windowPos and af.windowSize, it automatically makes af.wakeUpLine equal to the number of the second row below the bottom of the array. If af.windowPos and af.windowSize are such that the bottom of the array is outside the frame, the value given to af.wakeUpLine by AF_SetSize is greater than frame height, i.e. invalid. It must be changed to something less than the frame height, otherwise AF does not work.
12	zoneWeights	ulong	0xFFFFFFFF	RW	Weights of the AF windows or zones. Bits [1:0] of this variable represent the weight of window W11, bits [3:2] the weight of W12, and so on to bits [31:30] that represent the weight of W44. Since each weight is represented by just 2 bits, it is allowed to have only 4 values, 0, 1/3, 2/3 or 1. Value stored in each 2 bits equals window weight times 3, so the value of 1 signifies the weight of 1/3, 2 stands for 2/3, and 3 for 1.
16	distanceWeight	uint	0xFF	RW	Reserved.

Table 15: Driver Variables-Auto Fo

Off	Name	Type	Default	RW	Description
17	bestPosition	uchar	0	RW	This variable is used in 3 different ways depending on values of bits 6 and 7 of af.mode. When bit 7 equals 1 (in manual lens control mode), the position of AF lens can be changed by changing the value of af.bestPosition, which is interpreted by the AF driver as logical lens position desired by its user. The AF driver reads af.bestPosition once every frame, and if it differs from current logical lens position (afm.curPos), the AF driver gives the AFM driver a command to make these variables equal by moving the lens. Physical movement of the lens corresponding to the change of afm.curPos to af.bestPosition always takes some time, during which it is best not to change the value of af.bestPosition to avoid possible errors. When af.mode bits [7:6] are both 0, af.bestPosition serves to store AF algorithm output rather than user input. After both first and second scan, the AF algorithm outputs to this variable the offset of programmable logical lens position found to be best relative to the start position of the scan. In other words, after each scan, the best lens position found is af.positions[af.initPos+af.bestPosition]. When af.mode bit 7 is 0 and bit 6 is 1 (creep compensation mode is enabled) af.bestPosition is used during lens repositioning triggered by setting bit 1 of af.mode to 1. It is used to store the desired final lens position, which is assumed to be the same as afm.curPos before the repositioning. As a result after every successful repositioning, af.bestPosition equals afm.curPos.
18	shaTH	uchar	10	RW	Sharpness score variability threshold. Only AF windows whose min. and max. normalized sharpness scores satisfy the condition 1-(min.score/max.score) > = af.shaTH/256 are used to select best focus position.
19	positions[0]	uchar	0	RW	Programmable logical lens position 0
20	positions[1]	uchar	28	RW	Programmable logical lens position 1
21	positions[2]	uchar	56	RW	Programmable logical lens position 2
22	positions[3]	uchar	85	RW	Programmable logical lens position 3
23	positions[4]	uchar	113	RW	Programmable logical lens position 4
24	positions[5]	uchar	141	RW	Programmable logical lens position 5
25	positions[6]	uchar	170	RW	Programmable logical lens position 6
26	positions[7]	uchar	198	RW	Programmable logical lens position 7
27	positions[8]	uchar	226	RW	Programmable logical lens position 8
28	positions[9]	uchar	255	RW	Programmable logical lens position 9
29	positions[10]	uchar	27	RW	Programmable logical lens position 10
30	positions[11]	uchar	55	RW	Programmable logical lens position 11
31	positions[12]	uchar	84	RW	Programmable logical lens position 12
32	positions[13]	uchar	112	RW	Programmable logical lens position 13
33	positions[14]	uchar	140	RW	Programmable logical lens position 14
34	positions[15]	uchar	169	RW	Programmable logical lens position 15
35	positions[16]	uchar	197	RW	Programmable logical lens position 16
36	positions[17]	uchar	225	RW	Programmable logical lens position 17
37	positions[18]	uchar	254	RW	Programmable logical lens position 18
38	positions[19]	uchar	26	RW	Programmable logical lens position 19

Auto Focus Mechanics

Table 16: Driver Variables–Auto Focus Mechanics Driver (ID = 6)

Offs	Name	Type	Default[1]	RW	Description
0	vmt	void*	E0AE	RW	Pointer to the driver's virtual method table (VMT). The AFM driver includes a separate set of control methods for each supported type of auto focus mechanism (see afm.type below). Each set of methods is assessable via a separate VMT. Pointing afm.vmt to one of those VMTs either enables the AF driver to control the corresponding type of AF mechanism via the GPIO or takes away the control of the GPIO from the AF driver.
2	type	uchar	0	RW	Type of auto focus mechanism (lens actuator) used: 0—none, 1—helimorph, 2—stepper motor, 3—AD5398. At sequencer initialization, this variable is set to 0 and the afm.vmt is pointed to the default VMT of the AFM driver, which makes the GPIO inaccessible to the AF driver. Enabling the AF driver to control a lens actuator via the GPIO involves 2 steps. First, afm.type must be set to t+128, where t is 1, 2 or 3. Second, the sequencer must be given REFRESH command by setting seq.cmd to 5. The nonzero 7th bit in afm.type forces the sequencer to call AFM_Init function upon that command. The function makes afm.type equal to t and points afm.vmt to the VMT through which the AFM driver methods for controlling actuator type t can be called.
3	curPos	uchar	0	RW	Current logical position[2]
4	prePos	uchar	0	RW	Previous logical position
5	status	uchar	0	RW	Lens actuator status: Bit 0—0 if all is OK, 1 if the actuator reported an error Bit 1—0 if the lens is stationary, 1 if it is moving Bit 2—0 if last lens movement direction was forward (+), 1 if the direction was backward (-) Bits [4:3]—number of current stepper motor position if afm.type=2; otherwise unused. After sending a command to change lens position to the lens actuator, the AFM driver sets bit 1 of afm.status to 1. The value of the bit remains 1 until the AFM driver gets information that the lens is not moving (either has stopped at the desired new position or has failed to reach it). The AF driver waits through all times when the lens is moving by having afm.status updated once every frame, reading its bit 1 and immediately going back to sleep if it equals 1.
6	posMin	uchar	0	RW	Lower limit of physical position range[3]
7	posMax	uchar	0	RW	Upper limit of physical position range. Can be set below afm.posMin to swap forward (+) and backward (-) directions of lens movement.
8	posMacro	uchar	0	RW	Logical macro position

Table 16: Driver Variables–Auto Focus Mechanics Driver (ID = 6) (continued)

Offs	Name	Type	Default[1]	RW	Description
9	backlash	uchar	0	RW	Logical size of backlash-compensating step that the AF driver can optionally use in lens positioning after the first scan. If bits [7:6] of af.mode are set to 0, this positioning is done by moving the lens directly from the end position of the scan, af.positions[af.initPos+af.numSteps-1] to the logical position found best, af.position[af.initPos+af.bestPosition]. The direction of this move is opposite to the direction of the scan, and therefore the move may not bring the lens to its intended physical destination, unless its logical length is adjusted upward to compensate for lens actuator backlash. To make this adjustment, the AF driver subtracts afm.backlash from the value of af.positions[af.initPos+af.bestPosition] and gives the result to the AFM driver as the logical position to move the lens to. Negative results of the subtraction are replaced with 0. Subtracting afm.backlash makes sense only when af.positions[af.initPos+af.numSteps-1] > af.positions[af.initPos+af.bestPosition]; otherwise addition is required. Since the AF driver always subtracts afm.backlash, backlash compensation using this variable is not recommended after scans done in the negative direction (e.g. from logical position 255 to logical position 0).

Table 16: Driver Variables–Auto Focus Mechanics Driver (ID = 6) (continued)

Offs	Name	Type	Default[1]	RW	Description
10	custCtrl	uchar	0	RW	Custom controls: 1-bit option switches and fine-tuning parameters for actuator control methods. The function of different bits of this variable depends on the current value of afm.type. If afm.type = 1, then: Bit 0—selects the length of commands sent to HD80 helimorph driver by function AFM_SetPosHelimorph (0 - 2 bytes, 1 - 3 bytes including enable byte), Bit 1—selects one of 2 possible relations between the argument of the function AFM_SetPosHelimorph, bPos, and position byte sent to HD80 helimorph driver (0 means send bPos, 1 - send 255-bPos, to reverse the direction of lens movement), Bit 2—selects one of 2 positions that helimorph can assume upon command to exit standby (0—afm.posMin, 1—afm.posMax), Bits [7:3] - unused. If afm.type = 2, then: Bit 0—selects direction of lens motion if bit 1 is set to 1, Bit 1—determines how function AFM_SetPosStMotor interprets its 1—byte argument (0—as desired logical lens position, 1—as number of physical steps to make in the direction indicated by bit 0), Bit 2—enables periodic forcing of stepper-motor-driving outputs into calculated logical states (0—forcing disabled, outputs are only toggled as needed, 1— forcing enabled), Bit 3—when set to 1, enables powering stepper motor down after every movement (the motor is always powered up before movements, but powering down is optional), Bit 4—enables repositioning of stepper motor by function AFM_ResetStMotor upon command to enter standby (1—enable, 0—disable), Bit 5—enables repositioning of stepper motor by function AFM_ResetStMotor upon command to exit standby (1—enable, 0—disable), Bits [7:6]—allow one to slow down initial portions of stepper-motor-driving waveforms that cannot be entirely generated by MT9D111 waveform generator (higher value = slower waveforms). If afm.type = 0, afm.custCtrl is unused.
11	timer.vmt	void*	E9C6	RW	Pointer to timer VMT. Default timer VMT located in ROM contains pointers to the following public functions: AFM_Wait, AFM_TimerSetDelay, AFM_TimerSetTimeToMove, AFM_TimerIsStopped. The pointers are all of type void* and have the following names: pWait, pSetDelay, pSetTimeToMove, pTimerIsStopped.
13	timer.startTime	uint	0	RW	Timer start time.
15	timer.stopTime	uint	0	RW	Timer stop time.

Table 16: Driver Variables□Auto Focus Mechanics Driver (ID = 6) (continued)

Offs	Name	Type	Default[1]	RW	Description
17	timer.hiWordMclkFreq	uint	0	RW	Master clock frequency in Hz divided by 65536. Used to convert delay times in milliseconds (for example, the values of afm.timer.maxShortDelay) to corresponding counts of master clock cycles that can be programmed into the timer.
19	timer.maxShortDelay	uint	0	RW	Maximum expected duration of short lens move. Should be given in milliseconds. Used by the AFM driver function AFM_TimerSetTimeToMove to compute lens travel time estimates.
21	timer.maxLongDelay	uint	0	RW	Maximum expected duration of long lens move. Should be given in milliseconds. Used by the AFM driver function AFM_TimerSetTimeToMove to compute lens travel time estimates.
23	timer.maxQuickMove	uchar	0	RW	Maximum length of short lens move (or threshold between short and long moves). Used by the AFM driver function AFM_TimerSetTimeToMove to compute lens travel time estimates.

Table 16: Driver Variables–Auto Focus Mechanics Driver (ID = 6) (continued)

Offs	Name	Type	Default[1]	RW	Description
24	timer.config	uchar	0	RW	Bits [1:0] of this variable determine how afm.timer.maxShortDelay, afm.timer.maxLongDelay, and afm.timer.maxQuickMove are used to estimate duration of lens movements. Bits [7:2] are unused. If a command-driven lens actuator does not provide any feedback about its status after receiving a command to move an AF lens, the AFM driver must somehow predict how long the lens will be moving, to prevent the AF driver from collecting sharpness scores and issuing new commands during its movement. The need for predictions of lens travel time is satisfied rather inexpensively by the AFM driver function AFM_TimerSetTimeToMove, which takes as arguments 2 logical lens positions and estimates the time required to move the lens between them. The function can use 2 different estimation methods, both of which rely on 3 user-set parameters, afm.timer.maxShortDelay, afm.timer.maxLongDelay, and afm.timer.maxQuickMove, as a sole source of information about how fast the lens actuator moves the lens. The default method of piecewise linear estimation is used when bit 0 of afm.timer.config is cleared. Setting this bit to 1 enables the alternative bipolar method. The bipolar method is very simple: if the distance between the 2 logical positions given to AFM_TimerSetTimeToMove as arguments exceeds afm.timer.maxQuickMove, then afm.timer.maxLongDelay is selected as the proper lens travel time estimate. Otherwise, unless the 2 logical positions are the same, the estimate equals afm.timer.maxShortDelay. If the 2 positions are the same, the estimate should be 0, and indeed is 0 if bit 1 of afm.timer.config is cleared. However, if this bit is set to 1 and the positions are the same, the function AFM_TimerSetTimeToMove outputs afm.timer.maxShortDelay instead of 0.
25	si.vmt	void*	E9CE	RW	Pointer to serial interface VMT. Default serial interface VMT located in ROM contains pointers to the following public functions: AFM_SiSetActvFlag, AFM_SiSendByte, AFM_SiRecvByte. The pointers are all of type void* and have the following names: pSendCmd, pSetActvFlag, pSendByte, pRecvByte.
27	si.clkMask	uint	0	RW	Mask selecting one of GPIO pads as the clock line of dedicated two-wire serial interface between the MT9D111 and a lens actuator (for example, helimorph).
29	si.dataMask	uint	0	RW	Mask selecting one of GPIO pads as the data line of the dedicated two-wire serial interface to the lens actuator.

Table 16: Driver Variables–Auto Focus Mechanics Driver (ID = 6) (continued)

Offs	Name	Type	Default[1]	RW	Description
31	si.clkQtrPrd	uint	0	RW	Delay for slowing down serial interface transmissions. The period of serial interface clock is asymptotically proportional to si.clkQtrPrd.
33	si.needsAck	uchar	0	RW	Switch enabling detection of ACK bits from the lens actuator (0—disabled, 1—enabled).
34	si.slaveAddr	uchar	0	RW	Lens actuator address used in serial interface transmissions.
35	sm.enabMask	uint	0	RW	Mask selecting one of GPIO pads as stepper-motor-enabling output.
37	sm.drv0Mask	uchar	0	RW	Mask selecting one of GPIO pads (GPIO[1] by default) as first stepper driving output.
38	sm.drv1Mask	uchar	0	RW	Mask selecting one of GPIO pads (GPIO[3] by default) as the second stepper driving output.
39	sm.drv2Mask	uchar	0	RW	Mask selecting one of the GPIO pads as third stepper-motor-driving output
40	sm.drv3Mask	uchar	0	RW	Mask selecting one of the GPIO pads as fourth stepper-motor-driving output
41	sm.drvsQtrPrd	uchar	0	RW	Delay for lengthening the period of stepper motor driving waveforms. The number of master/GPIO clock cycles in this period asymptotically approaches 8 times the sm.drvsQtrPrd.
43	sm.drvsGenMode	uchar	0	RW	This variable tells the AFM driver how to program the GPIO to generate stepper motor driving waveforms. Bits [1:0]—size of smallest stepper motor move (0 – 4 steps, 1– 1 step, 2 – 2 steps) Bit 2—if 0, use the waveform generator in 8-bit counter mode, if 1, use it in 16-bit counter mode Bit 3—if 0, use clock divider 1, if 1, use clock divider 2 Bits [7:4]—clock divider setting (used by the AFM driver only if it is higher than the setting the driver has automatically calculated).
44	sm.piEnabMask	uint	0	RW	Mask selecting one of GPIO pads as photointerrupter-enabling output.
46	sm.piOutMask	uint	0	RW	Mask selecting one of GPIO pads as photointerrupter sensing input.
48	sm.piEdgeOffset	uchar	0	RW	Distance (in units of smallest stepper motor move) between the position of photointerrupter signal edge and desired initial position of stepper motor.
49	sm.piConfig	uchar	0	RW	Photointerrupter (PI) configuration Bit 0—if 0, do not use PI, if 1, use it for initial stepper positioning Bit 1—if 0, PI signal edge is near logical position 0, if 1—near 255. Bit2—PI active state (0—low, 1—high). Bit 3—if 1, PI is in the active state at initial stepper position, if 0—it is not. Bit 4—if 0, initial logical stepper position is 0, if 1—255. Bits [7:5]—delay between powering up the PI and sensing its output the first time.

NOTE:

1. Shown default values correspond to afm.type = 0

2. When the AFM driver exchanges information about lens position and motion with the AF driver, it is done in terms of logical position and displacement. Logical position range is from 0 to 255, irrespective of the lens actuator used and the number of distinct physical positions it can put the lens in.
3. If the lens actuator is physically or logically incapable of putting the lens in 256 evenly spaced positions, the AFM driver must translate between the range of logical positions recognized by the AF driver and the set of physical positions that the actuator can assume. The positions in this set can usually be assigned integer numbers forming a contiguous range. The limits of this range are the minimum information needed by the AFM driver to convert the logical positions and displacements to their physical equivalents.

Context

Table 17: Driver Variables–Mode/Context Driver (ID = 7)

Offs	Name	Type	ASSOC.H/W Reg	RW	Default	Description
0	VMT Pointer	uint	local	RW	59871	Pointer to virtual method table.
2	context	uchar	local	RW	0	Current context (0 = A, 1 = B).
3	output width_A	uint	R0x16:1	RW	800	Output size of final image for context A. Must be equal to or smaller than crop dimension.
5	output height_A	uint	R0x17:1	RW	600	Output size of final image for context A. Must be equal to or smaller than crop dimension.
7	output width_B	uint	R0x16:1	RW	1600	Output size of final image for context B. Must be equal to or smaller than crop dimension.
9	output height_B	uint	R0x17:1	RW	1200	Output size of final image for context B. Must be equal to or smaller than crop dimension.
11	mode_config	uint	R0xA:2	RW	0x10	Bit 4—JPG bypass: context A. Bit 5—JPG bypass: context B. Set to "0" to enable JPEG.
13	PLL_lock_delay	uint	local	RW	200	Delay between PLL enable and start of microcontroller operation (no. of clock cycles x 100).
15	s_row_start_A	uint	R0x01:0	RW	28	First sensor-readout row (context A shadow register).
17	s_col_start_A	uint	R0x02:0	RW	60	First sensor-readout column (context A shadow register).
19	s_row_height_A	uint	R0x03:0	RW	1200	No. of sensor-readout rows (context A shadow register).
21	s_col_width_A	uint	R0x04:0	RW	1600	No. of sensor-readout columns (context A shadow register).
23	s_ext_delay_A	uint	R0x0B:0	RW	0	Extra sensor delay per frame (context A shadow register).
25	s_row_speed_A	uint	R0x0A:0	RW	1	Row speed (context A shadow register).
27	s_row_start_B	uint	R0x01:0	RW	28	First sensor-readout row (context B shadow register).
29	s_col_start_B	uint	R0x02:0	RW	60	First sensor-readout column (context B shadow register).
31	s_row_height_B	uint	R0x03:0	RW	1200	No. of sensor-readout rows (context B shadow register).
33	s_col_width_B	uint	R0x04:0	RW	1600	No. of sensor-readout columns (context B shadow register).
35	s_ext_delay_B	uint	R0x0B:0	RW	1050	Extra sensor delay per frame (context B shadow register).
37	s_row_speed_B	uint	R0x0A:0	RW	1	Row speed (context B shadow register).
39	crop_X0_A	uint	R0x11:1	RW	0	Lower-x decimator zoom window (context A shadow register).
41	crop_X1_A	uint	R0x12:1	RW	800	Upper-x decimator zoom window (context A shadow register).

Table 17: Driver Variables–Mode/Context Driver (ID = 7) (continued)

Offs	Name	Type	ASSOC.H/W Reg	RW	Default	Description
43	crop_Y0_A	uint	R0x13:1	RW	0	Lower-y decimator zoom window (context A shadow register).
45	crop_Y1_A	uint	IR0x14:1	/W	600	Upper-y decimator zoom window (context A shadow register).
47	dec_ctrl_A	uint	R0x15:1	R/W	0	Decimator control register (context A shadow register).
53	crop_X0_B	uint	R0x11:1	RW	0	Lower-x decimator zoom window (context B shadow register).
55	crop_X1_B	uint	R0x12:1	RW	1600	Upper-x decimator zoom window (context B shadow register)
57	crop_Y0_B	uint	R0x13:1	RW	0	Lower-y decimator zoom window (context B shadow register).
59	crop_Y1_B	uint	R0x14:1	RW	1200	Upper-y decimator zoom window (context B shadow register).
61	dec_ctrl_B	uint	R0x15:1	RW	0	Decimator control register (context B shadow register).
67	gam_cont_A	uchar	local	RW	0x42	Gamma and contrast settings (context A shadow register) Gamma Setting: Bits 0:2: 0—gamma = 1.0 1—gamma = 0.56 2—gamma = 0.45 3—use user-defined gamma table Contrast Setting: Bits 4:6: 0—no contrast increase (slope = 100%) 1—some contrast increase (slope = 125%) 2—more contrast increase (slope = 150%) 3—most contrast increase (slope = 175%) 4—noise-reduction contrast
68	gam_cont_B	uchar	local	RW	0x42	Gamma and contrast settings (context B shadow register) Gamma Setting: Bits 0:2: 0—gamma = 1.0 1—gamma = 0.56 2—gamma = 0.45 3—use user-defined gamma table Contrast Setting: Bits 4:6: 0—no contrast increase (slope = 100%) 1—some contrast increase (slope = 125%) 2—more contrast increase (slope = 150%) 3—most contrast increase (slope = 175%) 4—noise-reduction contrast
69	gamma_table_A_0	uchar	R0xB2:1[7:0]	RW	0	User-defined gamma table values (context A shadow register).
70	gamma_table_A_1	uchar	R0xB2:1[15:8]	RW	39	User-defined gamma table values (context A shadow register).
71	gamma_table_A_2	uchar	R0xB3:1[7:0]	RW	53	User-defined gamma table values (context A shadow register).
72	gamma_table_A_3	uchar	R0xB3:1[15:8]	RW	72	User-defined gamma table values (context A shadow register).

Table 17: Driver Variables–Mode/Context Driver (ID = 7) (continued)

Offs	Name	Type	ASSOC.H/W Reg	RW	Default	Description
73	gamma_table_A_4	uchar	R0xB4:1[7:0]	RW	99	User-defined gamma table values (context A shadow register).
74	gamma_table_A_5	uchar	R0xB4:1[15:8]	RW	119	User-defined gamma table values (context A shadow register).
75	gamma_table_A_6	uchar	R0xB5:1[7:0]	RW	136	User-defined gamma table values (context A shadow register).
76	gamma_table_A_7	uchar	R0xB5:1[15:8]	RW	150	User-defined gamma table values (context A shadow register).
77	gamma_table_A_8	uchar	R0xB6:1[7:0]	RW	163	User-defined gamma table values (context A shadow register).
78	gamma_table_A_9	uchar	R0xB6:1[15:8]	RW	175	User-defined gamma table values (context A shadow register).
79	gamma_table_A_10	uchar	R0xB7:1[7:0]	RW	186	User-defined gamma table values (context A shadow register).
80	gamma_table_A_11	uchar	R0xB7:1[15:8]	RW	196	User-defined gamma table values (context A shadow register).
81	gamma_table_A_12	uchar	R0xB8:1[7:0]	RW	206	User-defined gamma table values (context A shadow register).
82	gamma_table_A_13	uchar	R0xB8:1[15:8]	RW	215	User-defined gamma table values (context A shadow register).
83	gamma_table_A_14	uchar	R0xB9:1[7:0]	RW	224	User-defined gamma table values (context A shadow register).
84	gamma_table_A_15	uchar	R0xB9:1[15:8]	RW	232	User-defined gamma table values (context A shadow register).
85	gamma_table_A_16	uchar	R0xBA:1[7:0]	RW	240	User-defined gamma table values (context A shadow register).
86	gamma_table_A_17	uchar	R0xBA:1[15:8]	RW	248	User-defined gamma table values (context A shadow register).
87	gamma_table_A_18	uchar	R0xBB:1[7:0]	RW	255	User-defined gamma table values (context A shadow register).
88	gamma_table_B_0	uchar	R0xB2:1[7:0]	RW	0	User-defined gamma table values (context B shadow register).
89	gamma_table_B_1	uchar	R0xB2:1[15:8]	RW	39	User-defined gamma table values (context B shadow register).
90	gamma_table_B_2	uchar	R0xB3:1[7:0]	RW	53	User-defined gamma table values (context B shadow register).
91	gamma_table_B_3	uchar	R0xB3:1[15:8]	RW	72	User-defined gamma table values (context B shadow register).
92	gamma_table_B_4	uchar	R0xB4:1[7:0]	RW	99	User-defined gamma table values (context B shadow register).
93	gamma_table_B_5	uchar	R0xB4:1[15:8]	RW	119	User-defined gamma table values (context B shadow register).
94	gamma_table_B_6	uchar	R0xB5:1[7:0]	RW	136	User-defined gamma table values (context B shadow register).
95	gamma_table_B_7	uchar	R0xB5:1[15:8]	RW	150	User-defined gamma table values (context B shadow register).
96	gamma_table_B_8	uchar	R0xB6:1[7:0]	RW	163	User-defined gamma table values (context B shadow register).

Table 17: Driver Variables–Mode/Context Driver (ID = 7) (continued)

Offs	Name	Type	ASSOC.H/W Reg	RW	Default	Description
97	gamma_table_B_9	uchar	R0xB6:1[15:8]	RW	175	User-defined gamma table values (context B shadow register).
98	gamma_table_B_10	uchar	R0xB7:1[7:0]	RW	186	User-defined gamma table values (context B shadow register).
99	gamma_table_B_11	uchar	R0xB7:1[15:8]	RW	196	User-defined gamma table values (context B shadow register).
100	gamma_table_B_12	uchar	R0xB8:1[7:0]	RW	206	User-defined gamma table values (context B shadow register).
101	gamma_table_B_13	uchar	R0xB8:1[15:8]	RW	215	User-defined gamma table values (context B shadow register).
102	gamma_table_B_14	uchar	R0xB9:1[7:0]	RW	224	User-defined gamma table values (context B shadow register).
103	gamma_table_B_15	uchar	R0xB9:1[15:8]	RW	232	User-defined gamma table values (context B shadow register).
104	gamma_table_B_16	uchar	R0xBA:1[7:0]	RW	240	User-defined gamma table values (context B shadow register).
105	gamma_table_B_17	uchar	R0xBA:1[15:8]	RW	248	User-defined gamma table values (context B shadow register).
106	gamma_table_B_18	uchar	R0xBB:1[7:0]	RW	255	User-defined gamma table values (context B shadow register).
107	FIFO_config0_A	uint	R0x0D:2	RW	0x0027	FIFO Buffer configuration 0 (context A shadow register).
109	FIFO_config1_A	uint	R0x0E:2	RW	0x0002	FIFO Buffer configuration 1 (context A shadow register).
111	FIFO_config2_A	uchar	R0x0F:2	RW	0x21	FIFO Buffer configuration 2 (context A shadow register).
112	FIFO_len_timing_A	uint	R0x12:2	RW	0x0606	FIFO spoof frame line timing (context B shadow register).
114	FIFO_config0_B	uint	R0x0D:2	RW	0x0067	FIFO Buffer configuration 0 (context B shadow register).
116	FIFO_config1_B	uint	R0x0E:2	RW	0x050A	FIFO Buffer configuration 1 (context B shadow register).
118	FIFO_config2_B	uchar	R0x0F:2	RW	0x0003	FIFO Buffer configuration 2 (context B shadow register).
119	FIFO_len_timing_B	uint	R0x12:2	RW	0x0606	FIFO spoof frame line timing (context B shadow register).
121	spoof_width_B	uint	R0x10:2*	RW	1600	FIFO spoof frame line timing (context B shadow register). *If in uncompressed mode, 2x this value is uploaded to R0x10.
123	spoof_height_B	uint	R0x11:2	RW	600	FIFO spoof frame line timing (context B shadow register).
125	out_format_A	uchar	R0x97:1	RW	0	Output Format Config. (context A shadow register).
126	out_format_B	uchar	R0x97:1	RW	0	Output Format Config. (context B shadow register).
127	spec_effects_A	uint	R0xA4:1	RW	0x6440	Special effects selection (context A shadow register).

Table 17: Driver Variables–Mode/Context Driver (ID = 7) (continued)

Offs	Name	Type	ASSOC.H/W Reg	RW	Default	Description
129	spec_effects_B	uint	R0xA4:1	RW	0x6440	Special effects selection (context B shadow register).
131	y_rgb_offset_A	uchar	R0xBF:1	RW	0	Y/RGB Offset setting (context A shadow register).
132	y_rgb_offset_B	uchar	R0xBF:1	RW	0	Y/RGB Offset setting (context B shadow register).

JPEG

Table 18: Driver Variables–JPEG Driver (ID = 9)

Offs	Name	Type	Def	RW	Description
0	vmt	void*	58451	RW	Reserved.
2	width	uint	1600	R	Image width (see mode.output_width).
4	height	uint	1200	R	Image height (see mode.output_height).
6	format	uchar	0	RW	Image format: 0—YCbCr 4:2:2 1—YCbCr 4:2:0 2—Monochrome
7	config	uchar	52	RW	Configuration and handshaking: Bit 0—if 1, video; if 0, still snapshot Bit 1—enable handshaking with host at every error frame Bit 2—enable retry after an unsuccessful encode or transfer Bit 3—host indicates it is ready for next frame Bit 4—enable scaled quantization table generation Bit 5—enable auto-select quantization table Bit 7:6—quantization table ID
8	restartInt	uint	0	RW	Restart marker interval: 0 = restart marker is disabled
10	qscale1	uchar	6	RW	Bit 6:0—scaling factor for first set of quantization tables Bit 7—1, new scaling factor value
11	qscale2	uchar	9	RW	Bit 6:0—scaling factor for second set of quantization tables Bit 7—if 1, new scaling factor value
12	qscale3	uchar	12	RW	Bit 6:0—scaling factor for third set of quantization tables Bit 7—if 1, new scaling factor value
13	timeoutFrames	uchar	10	RW	Number of frames to time out when host is not responding (setting bit 3 of config) to an unsuccessful JPEG frame while bit 1 of config is set.
14	state	uchar	0	R	JPEG driver state.
15	dataLengthMSB	uchar		R	Bit [23:16] of previous frame JPEG data length.
16	dataLengthLSBs	uint		R	Bit [15:0] of previous frame JPEG data length.

Histogram

Table 19: Driver Variables–Histogram Driver (ID = 11)

Offs	Name	Type	Def	RW	Description
0	vmt	void*	59880	RW	Reserved.
2	DlevelBufferSpeed	uchar	8	RW	Response speed, 1–32; 32-maximum speed.
3	scaleGFactor	uchar	1	RW	Scale factor for histogram window size.
4	maxDLevel	uchar	64	RW	Maximum subtracted offset. Set to “0” to disable subtraction.
5	percent	uchar	0	RW	Percent of pixels to keep black; 10-bits data / 4. Setting >0 clips black.
6	lowerLimit1	uchar	0	RW	Offset for bin 0, divided by 4 on 10-bit scale.
7	binSize1	uchar	2	RW	Bin width, 0–4LSB, 1–8LSB, 2–16LSB, 7–512LSB on a 10-bit scale.
8	lowerLimit2	uchar	192	RW	Offset for bin 0, divided by 4 on 10-bit scale.
9	binSize2	uchar	4	RW	Bin width, 0–4LSB, 1–8LSB, 2–16LSB, 7–512LSB on a 10-bit scale.
10	Dlevel	uchar		R	Current subtracted offset.
11	DLevel_buf	uint		R	Buffered current offset.
13	factorHi	uchar	10	RW	Factor of overexposure compensation for mettering mode.
14	percentHi	uchar	0	RW	Highlight clipping 255 - 100%

MCU Register List and Memory Map

Memory Map

Table 20: Memory Map

Address	Type	Description
0x8000-0xFFFF	ROM	System ROM, 32K 0xFFFE-0xFFFF—Reset vector 0xFFFC-0xFFFD—Driver table pointer 0xFFF8-0xFFF9—Illegal opcode vector 0xFFF6-0xFFF7—Software interrupt vector 0xFFF4-0xFFF5—XIRQ/Watchdog vector
0x1000-0x10FF	SFR	See Table 21, Special Function Register List.
0x0400-0x07FF	RAM	User SRAM, 1K. This SRAM is not used by the ROM firmware. Use it to upload custom code.
0x0000-0x03FF	RAM	System SRAM, 1K. Contains stack and data used by ROM firmware. 0x0380-0x03FF—Stack 0x0000-0x00FF—System short data segment

Special Function Registers - System

Table 21: Special Function Register List

Name	Hex#	Description
Native Registers		
PACTL	0x1026	Pulse accumulator control [2]—IC4/OC5 Input capture 4/output compare 5 select
OPTION	0x1039	System configuration options register [5]— IRQE 0 = low level sensitive,1 = falling edge sensitive
HPRIO	0x103C	Highest priority I-Bit interrupt and miscellaneous register [5]—MDA mode select A bit [3:0]—PSEL[3:0] priority select bits
INIT	0x103D	RAM and I/O mapping register [3:0]—REG[3:0] RAM map position bits [7:4]—RAM[3:0] register block position bits
Math Coprocessor Registers		
CREG[31:0]	0x10C0–0x10C3	32-bit data register
ALUC	0x10C4	Arithmetic logic unit control register [7]—signed number enable [6]—division enable [5]—multiply with accumulated product enable [4]—division compensation for concatenated quotient enable [3]—function start trigger bit [2]—overflow interrupt enable [1]—divide by zero interrupt enable [0]—arithmetic operation completion interrupt enable
AREG[15:0]	0x10C5–0x10C6	16-bit data register
BREG	0x10C70–0x10C8	16-bit data register
ALUF	0x10C9	Arithmetic logic unit flag register [7]—negative result flag [6]—remainder zero flag [5:3]—always “0” [2]—overflow MSB on CREG detected flag [1]—divide by zero detected flag; cleared by writing a “1” [0]—arithmetic operation completed flag; cleared by writing a “1”
SLEEP REGISTERS		
Sleep	0x1040	Sleep register 1—“1” = GPIO wakeup 0—“1” = line counter wakeup; write “1” to clear Read this register to attempt sleeping
WakeupLineCnt	0x1042/3	Line Counter Wakeup (MSB) 15—“0” = wake up on positive edge/level 14 —“1” = wake up on level, “0”= on edge [10:0]—Line number The following values are special: 0x07FF—disable wakeup 0x07FE—wake up on SOC frame enable 0x07FD—wake up on sensor frame enable 0x07FC—wake up on JPEG frame enable 0x07FB—wake up on FIFO frame enable

Table 21: Special Function Register List (continued)

Name	Hex#	Description
LineCnt	0x1046/7	Line counter (read-only) Current line number incremented at every line end: —0 at frame start —0x07FF at frame end Address 0x1046 (MSB) must be read first.
ClockCnt	0x1048-B	Clock counter (read-only) Address 0x1048 (MSB) must be read first.
RestartCode	0x104C	Warm restart register Bits 2:0—Warm restart code Value of 7 indicates watchdog caused reset.
InfoCode	0x104D	Debug information, maps to two-wire serial interface R195[15:8].
Watchdog	0x104E	Watchdog register Bit 3—1 watchdog causes XIRQ, 0=reset Bits 2:0—0 disable, other-frame number that triggers watchdog (1 = next frame) Clear Sleep reg. to reset watchdog.
BootMode	0x104F	Boot mode, read-only. Maps to two-wire serial interface R195[7:0].
Register I/O Bus Master		
WRITE_IOPAGE	0x1060	Register page number to write [0]—page number (IFP/Sensor core)
WRITE_IOADR	0x1061	Register address to write
IODATA	0x1062/3	Data from read transaction (read) Data for write operation (write) Write data to this register to initiate a register bus write transaction.
READ_IOPAGE	0x1064	Register page number to read Bit[0] page number (IFP/Sensor core)
READ_IOADR	0x1065	Register address for read transaction Write to initiate read transaction
IOSTATUS	0x1066	I/O bus transaction status (read-only) [7]—1 = write busy [6]—1 = read busy Read 0 to ensure I/O transaction is complete.
VARIABLE ACCESS (SLAVE DMA)		
R200	0x1050/1	Driver table pointer

Math Co-Processor Operations

Table 22: Math Co-Processor

7	6	5	4	Function	Start Triggers On
0	0	0	X	Unsigned MUL	Write to BREG LOW byte
1	0	0	X	Signed MUL	Write to BREG LOW byte
0	0	1	X	Unsigned MAC	Write to BREG LOW byte
1	0	1	X	Signed MAC	Write to BREG LOW byte
0	1	0	X	Unsigned IDIV	Write to AREG LOW byte or set TRG
1	1	0	0	Signed IDIV	Write to AREG LOW byte or set TRG
1	1	0	1	Signed IDIV DCC	Write to AREG LOW byte or set TRG
0	1	1	X	Unsigned FDIV	Set TRG
1	1	1	0	Signed FDIV	Set TRG
1	1	1	1	Signed FDIV DCC	Set TRG

JPEG Indirect Registers

Special Function Registers - GPIO

Table 23: GPIO Registers

Register Name	Bits	Register Content/Function	Addr (Hex)	Default
GPIO_DATA_H	3:0	Shows the state of GPIO pads 11:8 and controls those configured as outputs. Bit b (b = 0, ...,3) shows/controls the state of the pad GPIO(b + 8). If certain GPIO pads are floating during STANDBY, the corresponding bits should be tied to "0" converting the pads to output to reduce standby current. See technical note TN0934 "Standby Sequence" for more details.	1070	N/A
GPIO_DATA_L	7:0	Shows the state of GPIO pads 7:0 and controls those configured as outputs. Bit b (b = 0, ...,7) shows/controls the state of the pad GPIO(b). If certain GPIO pads are floating during STANDBY, the corresponding bits should be tied to "0" converting the pads to output to reduce standby current. See technical note TN0934 "Standby Sequence" for more details.	1071	N/A
GPIO_OUTPUT_TOGGLE_H[1]	3:0	Writing "1" to bit b (b = 0,...,3) toggles the GPIO(b + 8) output.	1072	0
GPIO_OUTPUT _TOGGLE_L	7:0	Writing "1" to bit b (b = 0,...,7) toggles the GPIO(b) output.	1073	0
GPIO_ OUTPUT _SET_H	3:0	Writing "1" to bit b sets the GPIO(b + 8) output HIGH.	1074	0
GPIO_ OUTPUT _SET_L	7:0	Writing "1" to bit b sets the GPIO(b) output HIGH.	1075	0
GPIO_ OUTPUT _CLEAR_H	3:0	Writing "1" to bit b sets the GPIO(b+8) output LOW.	1076	0
GPIO_ OUTPUT _CLEAR_L	7:0	Writing "1" to bit b sets the GPIO(b) output LOW.	1077	0
GPIO_DIR_H	3:0	Controls the direction of GPIO pads 11:8. If bit b (b = 0,...,3) is set to "1," the pad GPIO(b + 8) is an input; otherwise it is an output. Upon power-up or reset all GPIO pads become inputs. If certain GPIO pads are floating during STANDBY, the corresponding bits should be tied to "0" converting the pads to output to reduce standby current. See technical note TN0934 "Standby Sequence" for more details.	1078	0x0F
GPIO_DIR_L	7:0	Controls the direction of GPIO pads 7:0. If bit b (b = 0,...,7) is set to "1," the pad GPIO(b) is an input; otherwise it is an output. Upon power-up or reset all GPIO pads become inputs. If certain GPIO pads are floating during STANDBY, the corresponding bits should be tied to "0" converting the pads to output to reduce standby current. See technical note TN0934 "Standby Sequence" for more details.	1079	0xFF
GPIO_DIR_REVERSE_H	3:0	Writing "1" to bit b (b = 0,..,3) reverses the direction of GPIO(b + 8).	107A	0
GPIO_DIR_REVERSE_L	7:0	Writing "1" to bit b (b = 0,...,7) reverses the direction of GPIO(b).	107B	0
GPIO_DIR_IN_H	3:0	Writing "1" to bit b configures the pad GPIO(b + 8) as an input.	107C	0
GPIO_DIR_IN_L	7:0	Writing "1" to bit b configures the pad GPIO(b) as an input.	107D	0
GPIO_DIR_OUT_H	3:0	Writing "1" to bit b configures the pad GPIO(b + 8) as an output.	107E	0
GPIO_DIR_OUT_L	7:0	Writing "1" to bit b configures the pad GPIO(b) as an output.	107F	0
GPIO_WG_T01	7:0	First subperiod[2] of waveform output at GPIO1 (in 8-bit counter mode) or bits 15:8 of first subperiod of waveform output at GPIO0 (in 16-bit counter mode).	1080	0
GPIO_WG_T00	7:0	First subperiod of waveform generated at GPIO0 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1081	0

Table 23: GPIO Registers (continued)

Register Name		Register Content/Function	Addr (Hex)	Default
GPIO_WG_T11	7:0	Second subperiod of waveform output at GPIO1 (in 8-bit counter mode) or bits 15:8 of second subperiod of the waveform output at GPIO0 (in 16-bit counter mode).	1082	0
GPIO_WG_T10	7:0	Second subperiod of waveform generated at GPIO0 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1083	0
GPIO_WG_T21	7:0	Third subperiod of waveform output at GPIO1 (in 8-bit counter mode) or bits 15:8 of third subperiod of waveform output at GPIO0 (in 16-bit counter mode).	1084	0
GPIO_WG_T20	7:0	Third subperiod of waveform generated at GPIO0 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1085	0
GPIO_WG_T31	7:0	Fourth subperiod of waveform output at GPIO1 (in 8-bit counter mode) or bits 15:8 of fourth subperiod of waveform output at GPIO0 (in 16-bit counter mode).	1086	0
GPIO_WG_T30	7:0	Fourth subperiod of waveform generated at GPIO0 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1087	0
GPIO_WG_T41	7:0	Fifth subperiod of waveform generated at GPIO1 (in 8-bit counter mode) or bits 15:8 of 5th subperiod of waveform output at GPIO0 (in 16-bit counter mode).	1088	0
GPIO_WG_T40	7:0	Fifth subperiod of waveform generated at GPIO0 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1089	0
GPIO_WG_N1	7:0	Writing to this register sets the duration of waveform generated at GPIO1 (in 8-bit counter mode) or bits 15:8 of the duration of waveform generated at GPIO0 (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing “0“makes the duration infinite. When read, this registers returns the number of waveform periods left to be generated at GPIO1 (in 8-bit counter mode) or bits 15:8 of the number remaining at GPIO0 (in 16-bit counter mode). To get all 16 bits of the latter number correct, one must read this register before GPIO_WG_N0.	108A	0
GPIO_WG_N0	7:0	Writing to this register sets the duration of waveform generated at GPIO0 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing “0” makes the duration infinite. When read, this register returns the number of waveform periods left to be generated at GPIO0 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	108B	0
GPIO_WG_T03	7:0	First subperiod of waveform generated at GPIO3 (in 8-bit counter mode) or bits 15:8 of first subperiod of waveform output at GPIO2 (in 16-bit counter mode).	108C	0
GPIO_WG_T02	7:0	First subperiod of waveform generated at GPIO2 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	108D	0
GPIO_WG_T13	7:0	Second subperiod of waveform output at GPIO3 (in 8-bit counter mode) or bits 15:8 of second subperiod of waveform output at GPIO2 (in 16-bit counter mode).	108E	0
GPIO_WG_T12	7:0	Second subperiod of waveform generated at GPIO2 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	108F	0

Table 23: GPIO Registers (continued)

Register Name	Bits	Register Content/Function	Addr (Hex)	Default
GPIO_WG_T23	7:0	Third subperiod of waveform generated at GPIO3 (in 8-bit counter mode) or bits 15:8 of third subperiod of waveform output at GPIO2 (in 16-bit counter mode).	1090	0
GPIO_WG_T22	7:0	Third subperiod of waveform generated at GPIO2 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1091	0
GPIO_WG_T33	7:0	Fourth subperiod of waveform output at GPIO3 (in 8-bit counter mode) or bits 15:8 of fourth subperiod of waveform output at GPIO2 (in 16-bit counter mode).	1092	0
GPIO_WG_T32	7:0	Fourth subperiod of waveform generated at GPIO2 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1093	0
GPIO_WG_T43	7:0	Fifth subperiod of waveform generated at GPIO3 (in 8-bit counter mode) or bits 15:8 of fifth subperiod of waveform output at GPIO2 (in 16-bit counter mode).	1094	0
GPIO_WG_T42	7:0	Fifth subperiod of waveform generated at GPIO2 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1095	0
GPIO_WG_N3	7:0	Writing to this register sets the duration of waveform generated at GPIO3 (in 8-bit counter mode) or bits 15:8 of the duration of waveform generated at GPIO2 (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing “0” makes the duration infinite. When read, this registers returns the number of waveform periods left to be generated at GPIO3 (in 8-bit counter mode) or bits 15:8 of the number remaining at GPIO2 (in 16-bit counter mode). To get all 16 bits of the latter number right, one must read this register before GPIO_WG_N2.	1096	0
GPIO_WG_N2	7:0	Writing to this register sets the duration of waveform generated at GPIO2 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing “0” makes the duration infinite. When read, this register returns the number of waveform periods left to be generated at GPIO2 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1097	0
GPIO_WG_T05	7:0	First subperiod of waveform generated at GPIO5 (in 8-bit counter mode) or bits 15:8 of first subperiod of waveform output at GPIO4 (in 16-bit counter mode).	1098	0
GPIO_WG_T04	7:0	First subperiod of waveform generated at GPIO4 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	1099	0
GPIO_WG_T15	7:0	Second subperiod of waveform output at GPIO5 (in 8-bit counter mode) or bits 15:8 of second subperiod of waveform output at GPIO4 (in 16-bit counter mode).	109A	0
GPIO_WG_T14	7:0	Second subperiod of waveform generated at GPIO4 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	109B	0
GPIO_WG_T25	7:0	Third subperiod of waveform output at GPIO5 (in 8-bit counter mode) or bits 15:8 of third subperiod of waveform output at GPIO4 (in 16-bit counter mode).	109C	0
GPIO_WG_T24	7:0	Third subperiod of waveform generated at GPIO4 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	109D	0

Table 23: GPIO Registers (continued)

Register Name	Bits	Register Content/Function	Addr (Hex)	Default
GPIO_WG_T35	7:0	Fourth subperiod of waveform output at GPIO5 (in 8-bit counter mode) or bits 15:8 of fourth subperiod of waveform output at GPIO4 (in 16-bit counter mode).	109E	0
GPIO_WG_T34	7:0	Fourth subperiod of waveform generated at GPIO4 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	109F	0
GPIO_WG_T45	7:0	Fifth subperiod of waveform output at GPIO5 (in 8-bit counter mode) or bits 15:8 of fifth subperiod of waveform output at GPIO4 (in 16-bit counter mode).	10A0	0
GPIO_WG_T44	7:0	Fifth subperiod of waveform generated at GPIO4 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10A1	0
GPIO_WG_N5	7:0	Writing to this register sets the duration of waveform generated at GPIO5 (in 8-bit counter mode) or bits 15:8 of the duration of waveform generated at GPIO4 (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing “0” makes the duration infinite. When read, this registers returns the number of waveform periods left to be generated at GPIO5 (in 8-bit counter mode) or bits 15:8 of the number remaining at GPIO4 (in 16-bit counter mode). To get all 16 bits of the latter number right, one must read this register before GPIO_WG_N4.	10A2	0
GPIO_WG_N4	7:0	Writing to this register sets the duration of waveform generated at GPIO4 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing “0” makes the duration infinite. When read, this registers returns the number of waveform periods left to be generated at GPIO4 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10A3	0
GPIO_WG_T07	7:0	First subperiod of waveform generated at GPIO7 (in 8-bit counter mode) or bits 15:8 of first subperiod of waveform output at GPIO6 (in 16-bit counter mode).	10A4	0
GPIO_WG_T06	7:0	First subperiod of waveform generated at GPIO6 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10A5	0
GPIO_WG_T17	7:0	Second subperiod of waveform output at GPIO7 (in 8-bit counter mode) or bits 15:8 of second subperiod of waveform output at GPIO6 (in 16-bit counter mode).	10A6	0
GPIO_WG_T16	7:0	Second subperiod of waveform generated at GPIO6 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10A7	0
GPIO_WG_T27	7:0	Third subperiod of waveform output at GPIO7 (in 8-bit counter mode) or bits 15:8 of third subperiod of waveform output at GPIO6 (in 16-bit counter mode).	10A8	0
GPIO_WG_T26	7:0	Third subperiod of waveform generated at GPIO6 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10A9	0
GPIO_WG_T37	7:0	Fourth subperiod of waveform output at GPIO7 (in 8-bit counter mode) or bits 15:8 of fourth subperiod of waveform output at GPIO6 (in 16-bit counter mode).	10AA	0
GPIO_WG_T36	7:0	Fourth subperiod of waveform generated at GPIO6 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10AB	0

Table 23: GPIO Registers (continued)

Register Name	Bits	Register Content/Function	Addr (Hex)	Default
GPIO_WG_T47	7:0	Fifth subperiod of waveform generated at GPIO7 (in 8-bit counter mode) or bits 15:8 of fifth subperiod of waveform output at GPIO6 (in 16-bit counter mode).	10AC	0
GPIO_WG_T46	7:0	Fifth subperiod of waveform generated at GPIO6 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10AD	0
GPIO_WG_N7	7:0	Writing to this register sets the duration of waveform generated at GPIO7 (in 8-bit counter mode) or bits 15:8 of the duration of waveform generated at GPIO6 (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing "0" makes the duration infinite. When read, this registers returns the number of waveform periods left to be generated at GPIO7 (in 8-bit counter mode) or bits 15:8 of the number remaining at GPIO6 (in 16-bit counter mode). To get all 16 bits of the latter number right, one must read this register	10AE	0
GPIO_WG_N6	7:0	Writing to this register sets the duration of waveform generated at GPIO6 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode). Finite duration is selected by writing the desired number of periods in the waveform. Writing "0" makes the duration infinite. When read, this register returns the number of waveform periods left to be generated at GPIO6 (in 8-bit counter mode) or bits 7:0 of the same (in 16-bit counter mode).	10AF	0
GPIO_WG_CONFIG	3:0	Setting/clearing bit b (b = 0,...,3) enables/disables waveform generation at the pads GPIO(2b) and GPIO(2b + 1).	10B0	0
	7:4	Clearing bit b (b = 4,...,7) enables the waveform generator to drive the pads GPIO(2b - 8) and GPIO(2b - 7) simultaneously. Counters needed to generate waveforms at these pads are put in the 8-bit mode. Setting bit b disconnects the waveform generator from the pad GPIO(2b - 7) and allows it to generate a waveform at the pad GPIO(2b - 8) using the 16-bit counter mode.		0
GPIO_WG_CHAIN	6:0	By setting bits 6:0 in this register, one can link waveforms generated at different pads into a chain. The end of the first waveform in the chain coincides with the start of the second waveform, and so on. Specifically, for b = 0, 2, 4, the following is true: Setting bit b forces the waveform generator to start driving GPIO(b) when it is done with GPIO(b + 1) (in 8-bit counter mode) or GPIO(b + 2) (in 16-bit counter mode). Setting bit (b + 1) forces the waveform generator in to start driving GPIO(b + 1) when done with GPIO(b + 2) (in 8-bit counter mode; the bit is ignored in 16-bit mode). In addition, setting bit 6 forces the waveforms generator to start driving GPIO6 when done with GPIO7 (in 8-bit counter mode; the bit is ignored in 16-bit mode).	10B1	0

Table 23: GPIO Registers (continued)

Register Name	Bits	Register Content/Function	Addr (Hex)	Default
GPIO_WG_CLKDIV	7:0	The waveform generator has two clock dividers that enable it to generate waveforms at vastly different paces. Each divider divides the frequency of the GPIO clock (typically 80 MHz) by a factor 2^{d+1}, where d is a 4-bit unsigned integer programmed into register GPIO_WG_CLKDIV. Bits 3:0 of this register hold d for divider 1. Bits 7:4 of this register hold d for divider 2.	10B2	0
GPIO_WG_CLKDIV_SEL	7:0	Cleared bit b (b = 0,…,7) tells the waveform generator to use clock divider 1 when generating waveforms at the pad GPIO(b). Set bits select clock divider 2 for the corresponding pads.	10B3	0
GPIO_WG_FRAME_SYNC	7:0	If bit b (b=0,…,7) in GPIO_WG_FRAME_SYNC is cleared, waveform generation at the GPIO(b) output starts/resumes when bit b in GPIO_WG_SUSPEND is cleared. Setting bit b in GPIO_WG_FRAME_SYNC changes the conditions that must be met for waveform generation at GPIO(b) can start/ resume. The clearing of bit b in GPIO_WG_SUSPEND is still necessary, but no longer sufficient. After that bit is cleared, the waveform generator restarts on each falling edge of FRAME_VALID.	10B4	0
GPIO_WG_RESET	7:0	Setting bit b (b = 0,…,7) stops any ongoing waveform generation at GPIO(b), and resets all counters used in it. The bit must be cleared before waveform generation can resume.	10B5	0
GPIO_WG_SUSPEND	7:0	Setting bit b (b = 0,…,7) suspends waveform generation at the pad GPIO(b). Clearing the bit restarts it.	10B6	0
GPIO_NS_TYPE	7:0	Setting bit b (b = 0,…,7) enables a notification signal (NS) at the end of waveform generation at GPIO(b). Clearing the bit b enables a NS on next transition at GPIO(b) whose sign matches the sign indicated by bit b in GPIO_NS_EDGE_L. The pad may be configured as an output or input and the transition may be caused by the waveform generator, writing to GPIO_DATA_L or external forcing.	10B8	0
GPIO_NS_EDGE_H	3:0	Bit b (b = 0,…,3) selects the sign of transitions on pad GPIO(b + 8) that triggers notification signals. Setting the bit selects rising edges, clearing it, the falling edges.	10B9	0
GPIO_NS_EDGE_L	7:0	Bit b (b = 0,…,7) selects the sign of transitions on pad GPIO(b) that triggers notification signals. Setting the bit selects rising edges, clearing it, the falling edges.	10BA	0
GPIO_NS_MASK_H	3:0	Setting bit b masks all notification signals caused by events on pad GPIO(b+8). Masked signals do not cause the microcontroller to wake up. Clearing the bit enables waking up.	10BB	0x0F
GPIO_NS_MASK_L	7:0	Setting bit b (b = 0,…,7) masks all notification signals caused by events on pad GPIO(b). Masked signals do not cause micro-controller to wake up. Clearing the bit enables waking up.	10BC	0xFF
GPIO_WG_STROBE_SYNC	7:0	If bit b (b = 0,…,7) in GPIO_WG_STROBE_SYNC is cleared, waveform generation at GPIO(b) output starts/resumes when bit b in 4 Setting bit b in GPIO_WG_STROBE_SYNC changes the conditions that must be met for waveform generation on GPIO(b) can start/ resume. The clearing of bit b in GPIO_WG_SUSPEND is still necessary, but no longer sufficient. After that bit is cleared, the waveform generator restarts on each rising edge of STROBE.	10BD	0

Table 23: GPIO Registers (continued)

Register Name	Bits	Register Content/Function	Addr (Hex)	Default
GPIO_NS_STATUS_H	3:0	When bit b (b = 0,…,3) is set, it signals that some event on pad GPIO(b+8) caused a notification signal. Writing "1" to the bit to clears it.	10BE	0
GPIO_NS_STATUS_L	7:0	When bit b (b = 0,…,7) is set, it signals that some event on pad GPIO(b) caused a notification signal. Writing "1" to the bit to clears it.	10BF	0

Notes: 1. Registers marked by light gray shading are write-only, i.e., do not give any information on GPIO status when read.
2. The term "subperiod" refers to the time intervals labeled with T in Figure 46 on page 173. The GPIO generates periodic waveforms like those shown there. One period of a waveform consists of 1 to 5 nonzero subperiods. The GPIO_WG_T* registers actually hold counter settings that have to be converted to time units to obtain true length of the subperiods. Please see "Waveform Programming" on page 163 for a more complete explanation.

Output Format and Timing

YUV/RGB Uncompressed Output

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

Figure 6: Timing of Decimated Uncompressed Output Bypassing the FIFO

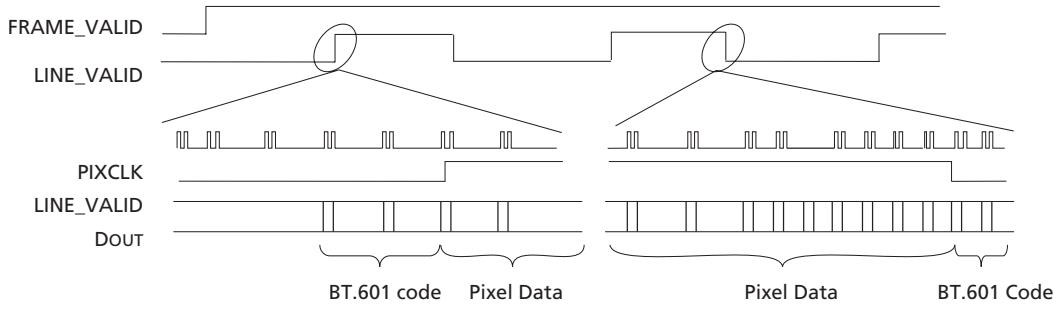
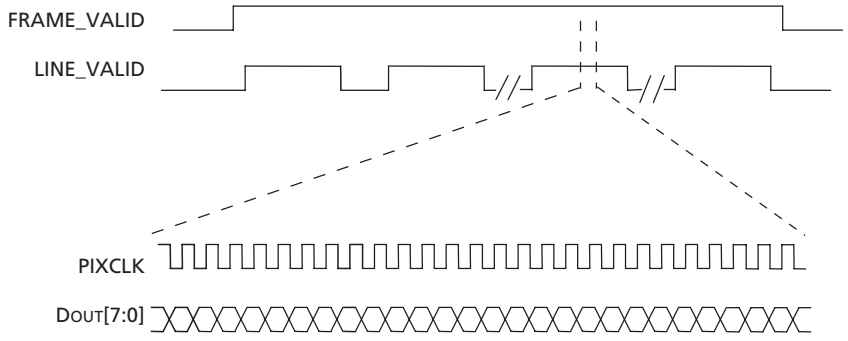


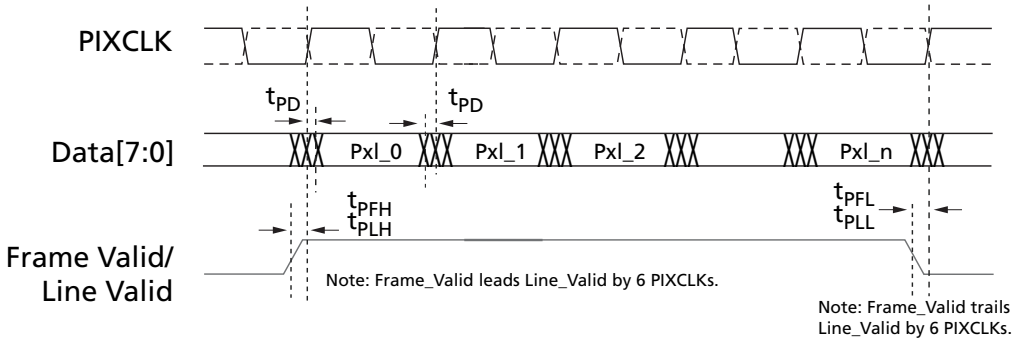
Figure 7 depicts the output timing of uncompressed YUV/RGB when a decimated data stream is equalized by buffering or when no decimation takes place. The pixel clock frequency remains constant during each LINE_VALID high period. Decimated data are output at a lower frequency than full size frames, which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

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



Timing details of uncompressed YUV/RGB output are shown in Figure 8 on page 111.

Figure 8: Details of Uncompressed YUV/RGB Output Timing



Symbol	Definition	Conditions	MIN	MAX	Units
f_{PIXCLK}	PIXCLK frequency	Default		80	MHz
t_{PD}	PIXCLK to data valid	Default	-3	3	ns
t_{PFH}	PIXCLK to FV high	Default	-3	3	ns
t_{PLH}	PIXCLK to LV high	Default	-3	3	ns
t_{PFL}	PIXCLK to FV low	Default	-3	3	ns
t_{PLL}	PIXCLK to LV low	Default	-3	3	ns

Uncompressed YUV/RGB Data Ordering

The MT9D111 supports swapping YCrCb mode, as illustrated in Table 24.

Table 24: YCrCb Output Data Ordering

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

The RGB output data ordering in default mode is shown in Table 25. The odd and even bytes are swapped when luma/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	$D_7D_6D_5D_4D_3D_2D_1D_0$
RGB 565	Odd	$R_7R_6R_5R_4R_3G_7G_6G_5$
	Even	$G_4G_3G_2B_7B_6B_5B_4B_3$
RGB 555	Odd	$0\ R_7R_6R_5R_4R_3G_7G_6$
	Even	$G_4G_3G_2B_7B_6B_5B_4B_3$
RGB 444x	Odd	$R_7R_6R_5R_4G_7G_6G_5G_4$
	Even	$B_7B_6B_5B_4\ 0\ 0\ 0\ 0$
RGB x444	Odd	$0\ 0\ 0\ 0\ R_7R6_6R_5R_4$
	Even	$G_7G_6G_5G_4B_7B_6B_5B_4$

Uncompressed 10-Bit Bypass Output

Raw 10-bit Bayer data from the sensor core can be output in bypass mode in two ways:

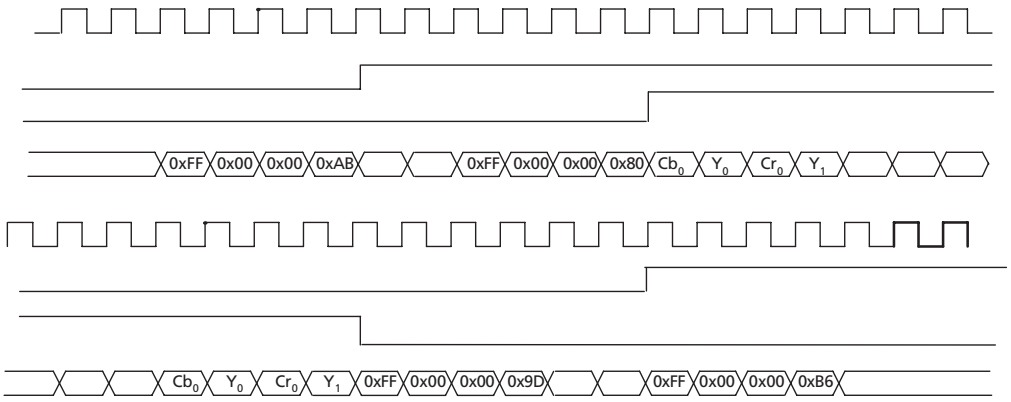
1. Using 10 data output pads (DOUT0–DOUT9), or
2. Using only 8 pads (DOUT0–DOUT7) and a special 8 + 2 data format, shown in Table 26.

The timing of 10-bit or 8-bit data stream output in the bypass mode is qualitatively the same as that depicted in Figure 7.

Table 26: 2-Byte RGB Format

Odd bytes	8 data bits	$D_9D_8D_7D_6D_5D_4D_3D_2$
Even bytes	2 data bits + 6 unused bits	0 0 0 0 0 0 D_1D_0

Figure 9: Example of Timing for Non-Decimated Uncompressed Output Bypassing Output FIFO



JPEG Compressed Output

JPEG compression of IFP output produces a data stream whose structure differs from that of an uncompressed YUV/RGB stream. Frames are no longer sequences of lines of constant length. This difference is reflected in the timing of the LINE_VALID signal. When JPEG compression is enabled, logical HIGHs on LINE_VALID do not correspond to image lines, but to variably sized packets of valid data. In other words, the LINE_VALID signal is in fact a DATA_VALID signal. It is a good analogy to compare the JPEG output of the MT9D111 to an 8-bit parallel data port wherein the LINE_VALID signal indicates valid data and the FRAME_VALID signal indicates frame timing.

The JPEG compressed data can be output either continuously or in blocks simulating image lines. The latter output scheme is intended to spoof a standard video pixel port connected to the MT9D111 and for that purpose treats JPEG entropy-coded segments as if they were standard video pixels. In the continuous output mode, JPEG output clock can be free running or gated. In all, three timing modes are available and are depicted in Figure 10 on page 114, Figure 11 on page 114, and Figure 12 on page 114. These timing diagrams are merely three typical examples of many variations of JPEG output. Description of output configuration register R13:2 in Table 7, "IFP Registers, Page 2," on page 52 provides more information on different output interface configuration options.

The "continuous" and spoof JPEG output modes differ primarily in how the LINE_VALID output is asserted. In the continuous mode, LINE_VALID is asserted only during output clock cycles containing valid JPEG data. The resulting LINE_VALID signal pattern is non-uniform and highly image dependent, reflecting the inherent nature of JPEG data stream. In the spoof mode, LINE_VALID is asserted and de-asserted in a more uniform pattern emulating uncompressed video output with horizontal blanking intervals. When LINE_VALID is de-asserted, available JPEG data are not output, but instead remain in the FIFO until LINE_VALID is asserted again. During the time when LINE_VALID is asserted, the output clock is gated off whenever there is no valid JPEG data in the FIFO.

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

The host processor configures the spoof pattern by programming the total number of LINE_VALID assertion intervals, as well as the number of output clock periods during and between LINE_VALID assertions. In other words, the host processor can define a temporal “frame” for JPEG output, preferably with “size” tailored to the expected JPEG file size. If this frame is too large for the total number of JPEG bytes actually produced, the MT9D111 either de-asserts FRAME_VALID or continues to pad unused “lines” with zeros until the end of the frame. If the frame is too small, the MT9D111 either continues to output the excess JPEG bytes until the entire JPEG compressed image is output or discards the excess JPEG bytes and sets an error flag in a status register accessible to the host processor.

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

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

Output clock speed can optionally be made to vary according to the fullness of the FIFO, to reduce the likelihood of FIFO overflow. When this option is enabled, the output clock switches at three fixed levels of FIFO fullness (25 percent, 50 percent and 75 percent) to a higher or lower frequency, depending on the direction of fullness change. The set of possible output clock frequencies is restricted by the fact that its period must be an integer multiple of the master clock period. The frequencies to be used are chosen by programming three output clock frequency divisors in registers R14:2 and R15:2. Divisor N1 is used if the FIFO is less than 50 percent full and last fullness threshold crossed has been 25 percent. When the FIFO reaches 50 percent and 75 percent fullness, the output clock switches to divisor N2 and N3, respectively. When the FIFO fullness level drops to 50 percent and 25 percent, the output clock is switched back to divisor N2 and N1, respectively.

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

Figure 10: Timing of JPEG Compressed Output in Free-Running Clock Mode

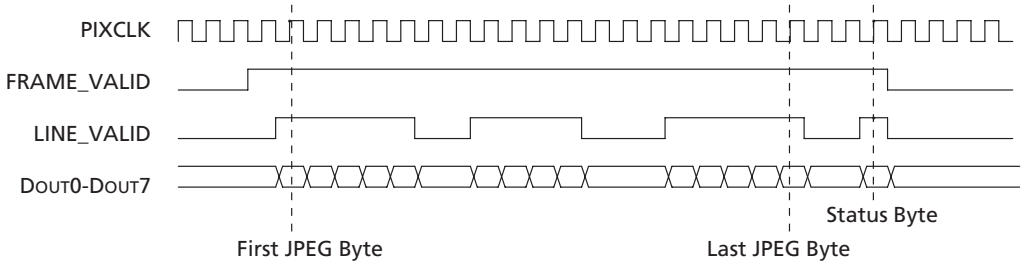


Figure 11: Timing of JPEG Compressed Output in Gated Clock Mode

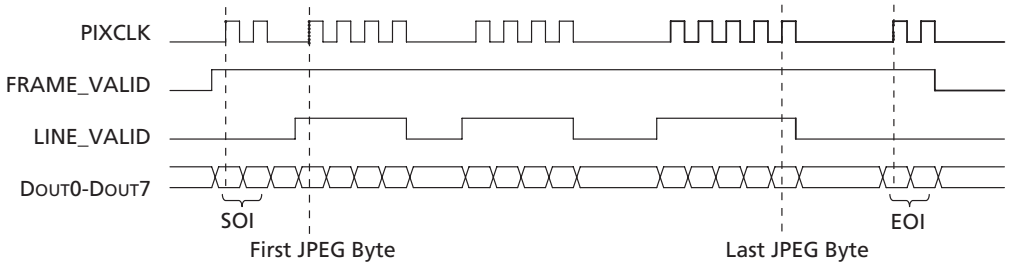
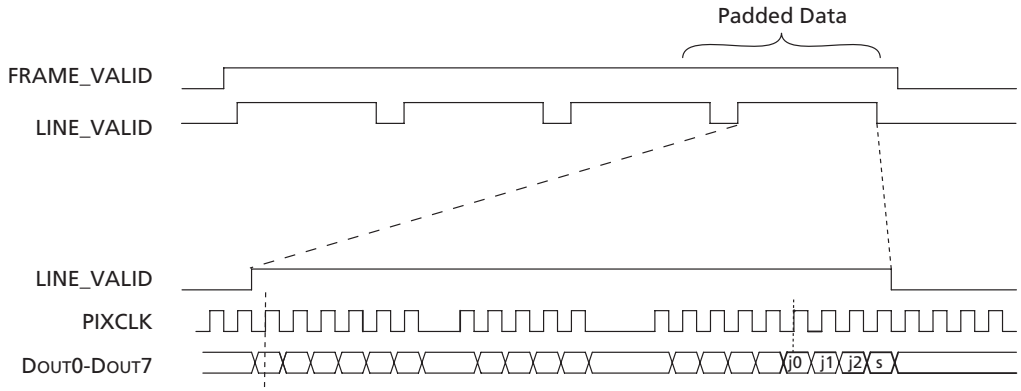


Figure 12: Timing of JPEG Compressed Output in Spoof Mode



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

Color Conversion Formulas

Y'Cb'Cr' ITU-R BT.601

A widely known color conversion standard. Note that 16 < Y_{601} < 235 and 16 < Cb, Cr < 240. 0 < = RGB < = 255.

Y_{601}' = Y'*219/256 + 16
Cr' = 0.713 (R' - Y')*224/256 + 128
Cb' = 0.564 (B' - Y')*224/256 + 128
where Y' = 0.299 R' + 0.587 G' + 0.114 B'

The reverse formulas to convert YCbCr into RGB, 0 < = RGB < = 255:
R' = 1.164(Y_{601}' - 16) + 1.596(Cr' - 128)
G' = 1.164(Y_{601}' - 16) - 0.813(Cr' - 128) - 0.391(Cb' - 128)
B' = 1.164(Y_{601}' - 16) + 2.018(Cb' - 128)

Y'U'V'

This conversion is BT 601 scaled to make YUV range from 0 through 255. This setting is recommended for JPEG encoding and is the most popular, although it is not well defined and often misused in Windows.

Y' = 0.299 R' + 0.587 G' + 0.114 B'
U' = 0.564 (B' - Y') + 128
V' = 0.713 (R' - Y') + 128

There is an option where 128 is not added to U'V'.

Y'Cb'Cr Using sRGB Formulas

The MT9D111 implements the sRGB standard. This option provides YCbCr coefficients for a correct 4:2:2 transmission. Note that 16 < Y60 1< 235, 16 < Cb, Cr < 240 and 0 < = RGB < = 255.

Y' = (0.2126*R' + 0.7152*G' + 0.0722*B')*219/256 + 16
Cb' = 0.5389*(B '- Y')*224/256 + 128
Cr' = 0.635*(R' - Y')*224/256 + 128

Y'U'V' Using sRGB Formulas

Similar to the previous set of formulas, but has YUV spanning a range of 0 through 255.

Y' = 0.2126*R' + 0.7152*G' + 0.0722*B'
U' = 0.5389*(B' - Y') + 128 = -0.1146*R - 0.3854*G + 0.5*B
V' = 0.635*(R' - Y') + 128 = 0.5*R - 0.4542*G - 0.0458*B

There is an option to disable adding 128 to U'V'. The reverse transform is as follows:

R' = Y + 1.5748*V
G' = Y - 0.1873*(U - 128) - 0.4681*(V - 128)
B' = Y + 1.8556*(U - 128)

Sensor Core

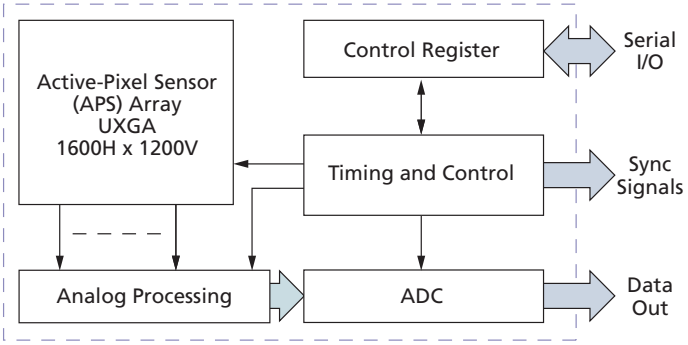
This section describes the sensor core. The core is based entirely on Micron's MT9D011 sensor.

The SOC firmware controls a key sensor core registers, such as exposure, window size, gains, and contexts. When firmware or MCU are disabled, the sensor core can be programmed directly.

Introduction

The sensor core is a progressive-scan sensor that generates a stream of pixel data qualified by LINE_VALID and FRAME_VALID signals. An on-chip PLL generates the master clock from an input clock of 6 MHz to 40 MHz. In default mode, the data rate (pixel clock) is the same as the master clock frequency, which means that one pixel is generated every master clock cycle. The sensor block diagram is shown in Figure 13.

Figure 13: Sensor Core Block Diagram



The core of the sensor is an active-pixel array. The timing and control circuitry sequences through the rows of the array, resetting and then reading each row. In the time interval between resetting a row and reading that row, the pixels in that row integrate incident light. The exposure is controlled by varying the time interval between reset and readout. After a row is read, the data from the columns is sequenced through an analog signal chain (providing offset correction and gain), and then through an ADC. The output from the ADC is a 10-bit value for each pixel in the array. The pixel array contains optically active and light-shielded "black" pixels. The black pixels are used to provide data for on-chip offset correction algorithms (black level control).

The sensor contains a set of 16-bit control and status registers that can be used to control many aspects of the sensor operations. These registers can be accessed through a two-wire serial interface. In this document, registers are specified either by name (Column Start) or by register address (R0x04:0). Fields within a register are specified by bit or by bit range (R0x20:0[0] or R0x0B:0[13 0]). The control and status registers are described in Table 5, "Sensor Register Description," on page 29.

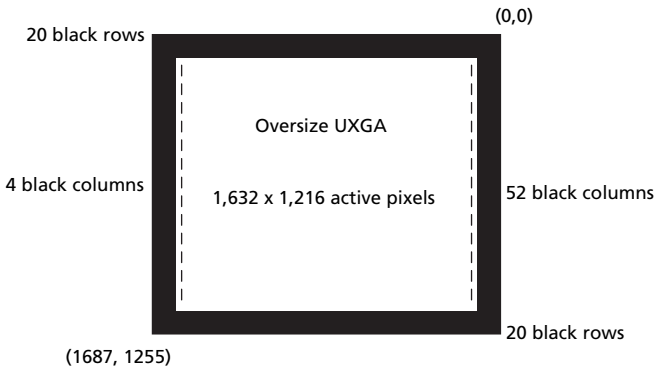
The output from the sensor is a Bayer pattern: alternate rows are a sequence of either green/red pixels or blue/green pixels. The offset and gain stages of the analog signal chain provide per-color control of the pixel data.

Pixel Array Structure

The sensor core pixel array is configured as 1,688 columns by 1,256 rows (shown in Figure 14). The first 52 columns and the first and the last 20 rows of pixels are optically black and are used for the automatic black level adjustment. The last 4 columns are also optically black.

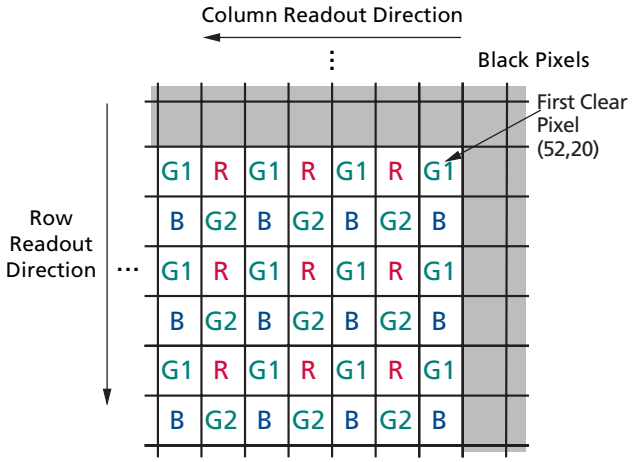
The optically active pixels are used as follows: In default mode a UXGA image (1,600 columns by 1,200 rows) is generated, starting at row 28, column 60. A 4-pixel boundary of active pixels can be enabled around the image to avoid boundary effects during color interpolation and correction. During mirrored readout, the region of active pixels used to generate the image is offset by 1 pixel in each mirrored direction so that the readout always starts on the same color pixel.

Figure 14: Pixel Array



The sensor core uses a Bayer color pattern, as shown in Figure 15. The even-numbered rows contain green and red color pixels; odd-numbered rows contain blue and green color pixels. Even-numbered columns contain green and blue color pixels; odd-numbered columns contain red and green color pixels. The color order is preserved during mirrored readout.

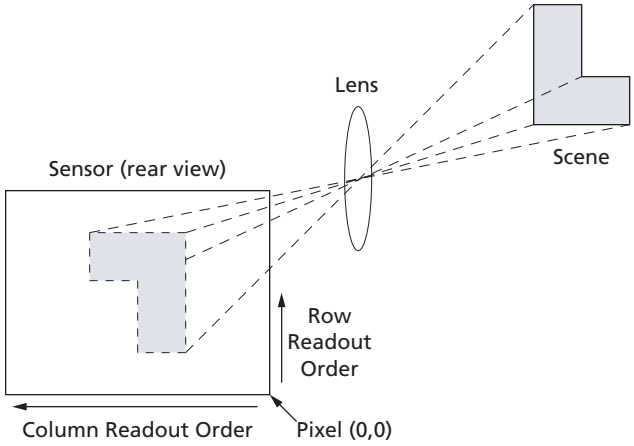
Figure 15: Pixel Color Pattern Detail (Top Right Corner)



Default Readout Order

By convention, the sensor core pixel array is shown with pixel (0,0) in the top right-hand corner (see Figure 15). This reflects the actual layout of the array on the die. When the sensor is imaging, the active surface of the sensor faces the scene as shown in Figure 16. When the image is read out of the sensor, it is read one row at a time, with the rows and columns sequenced as shown in Figure 14. By convention, data from the sensor is shown with the first pixel read out—pixel (52,20) in the case of the sensor core—in the top left-hand corner.

Figure 16: Imaging a Scene



Raw Data Format

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 17. The amount of horizontal blanking and vertical blanking is programmable. LINE_VALID is HIGH during the shaded region of the figure. FRAME_VALID timing is described in the next section.

Figure 17: Spatial Illustration of Image Readout

$P_{0,0}$ $P_{0,1}$ $P_{0,2}$ ..... $P_{0,n-1}$ $P_{0,n}$ $P_{1,0}$ $P_{1,1}$ $P_{1,2}$ ..... $P_{1,n-1}$ $P_{1,n}$ VALID IMAGE $P_{m-1,0}$ $P_{m-1,1}$ ..... $P_{m-1,n-1}$ $P_{m-1,n}$ $P_{m,0}$ $P_{m,1}$ ..... $P_{m,n-1}$ $P_{m,n}$	00 00 00 ..... 00 00 00 00 00 00 ..... 00 00 00 HORIZONTAL BLANKING 00 00 00 ..... 00 00 00 00 00 00 ..... 00 00 00
00 00 00 ..... 00 00 00 00 00 00 ..... 00 00 00 VERTICAL BLANKING 00 00 00 ..... 00 00 00 00 00 00 ..... 00 00 00	00 00 00 ..... 00 00 00 00 00 00 ..... 00 00 00 VERTICAL/HORIZONTAL BLANKING 00 00 00 ..... 00 00 00 00 00 00 ..... 00 00 00

Raw Data Timing

The sensor core output data is synchronized with the PIXCLK output. When LINE_VALID is HIGH, one pixel datum is output on the 10-bit DOUT output every PIXCLK period. By default, the PIXCLK signal runs at the same frequency as the master clock, and its rising edges occur one-half of a master clock period after transitions on LINE_VALID, FRAME_VALID, and DOUT (see Figure 18). This allows PIXCLK to be used as a clock to sample the data. PIXCLK is continuously enabled, even during the blanking period. The sensor core can be programmed to delay the PIXCLK edge relative to the DOUT transitions from 0 to 3.5 master clocks, in steps of one-half of a master clock. This can be achieved by programming the corresponding bits in R0x0A:0. The parameters P, A, and Q in Figure 19 are defined in Table 27 on page 120.

Figure 18: Pixel Data Timing Example

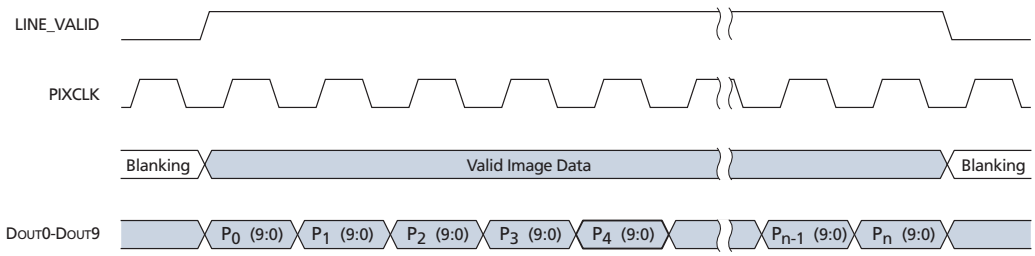
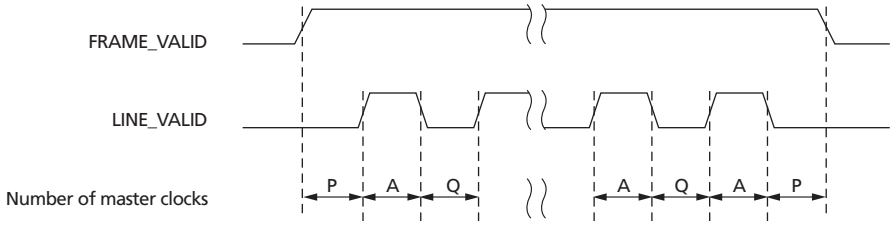


Figure 19: Row Timing and FRAME_VALID/LINE_VALID Signals



The sensor timing is shown in terms of pixel clock and master clock cycles (see Figure 18 on page 119). The recommended master clock frequency is 36 MHz. Increasing the integration time to more than one frame causes the frame time to be extended. The equations in Table assume integration time is less than the number of rows in a frame (R0x09:0 < R0x03:0/S + BORDER + VBLANK_REG). If this is not the case, the number of integration rows must be used instead to determine the frame time, as shown in Table 28.

Table 27: Frame Time

Parameter	Name	Equation	Default Timing at 36 MHz Dual ADC Mode
HBLANK_REG	Horizontal Blanking Register	R0x07:0 if R0xF2:0[0] = 0 R0x05:0 if R0xF2:0[0] = 1	0x15C = 348 pixels
VBLANK_REG	Vertical Blanking Register	R0x8:0 if R0xF2:0[1] = 0 R0x6:0 if R0xF2:0[1] = 1	0x20 = 32 rows
ADC_MODE	ADC mode	R0xF2:0[3] = 0: R0x20:0[10] R0xF2:0[3] = 1: R0x21:0[10]	
PIXCLK_PERIOD	Pixel clock period	ADC_MODE = 0: R0x0A:0[2:0] ADC_MODE = 1: R0x0A:0[2:0]*2	1 ADC_MODE: 55.556ns 2 ADC_MODE: 27.778ns
S	Skip Factor	For skip 2x mode: S = 2 For skip 4x mode: S = 4 For skip 8x mode: S = 8 For skip 16x mode: S = 16 otherwise, S = 1	1
A	Active Data Time	(R0x04:0/S) * PIXCLK_PERIOD	1,600 pixel clocks = 1,600 master = 44.44µs
P	Frame Start/End Blanking	6 * PIXCLK_PERIOD (can be controlled by R0x1F:0)	6 pixel clocks = 12 master = 0.166µs
Q	Horizontal Blanking	HBLANK_REG * PIXCLK_PERIOD	348 pixel clocks = 348 master = 9.667µs
A + Q	RowTime	((R0x04:0/S) + HBLANK_REG) * PIXCLK_PERIOD	1,948 pixel clocks = 1,948 master = 54.112µs
V	Vertical Blanking	VBLANK_REG * (A + Q) + (Q - 2*P)	62,672 pixel clocks = 62,672 master = 1.741ms
Nrows * (A + Q)	Frame Valid Time	(R0x03:0/S) * (A + Q) - (Q - 2*P)	2,337,264 pixel clocks = 2,337,264 master = 64.925ms
F	Total Frame Time	((R0x03:0/S) + VBLANK_REG) * (A + Q)	2,399,936 pixel clocks = 2,399,936 master = 66.665ms

Note: Skip factor should be multiplied by 2 if binning is enabled.

Table 28: Frame—Long Integration Time

Parameter	Name	Equation (Master Clock)
V′	Vertical Blanking (long integration time)	(R0x09:0 – (R0x03:0)/S) * (A + Q) + (Q - 2*P)
F′	Total Frame Time (long integration time)	(R0x09:0) * (A + Q)

Registers

Table 5, "Sensor Register Description," on page 29 provides a detailed description of the registers. Bit fields that are not identified in the table are read only.

Double-Buffered Registers

Some sensor settings cannot be changed during frame readout. For example, changing row width R0x03:0 part way through frame readout results in inconsistent LINE_VALID behavior. To avoid this, the sensor core double buffers many registers by implementing a "pending" and a "live" version. READs and WRITEs access the pending register. The live register controls the sensor operation.

The value in the pending register is transferred to a live register at a fixed point in the frame timing, called "frame start." Frame start is defined as the point at which the first dark row is read out. By default, this occurs 10 row times before FRAME_VALID goes HIGH. R0x22:0 enables the dark rows to be shown in the image, but this has no effect on the position of frame start.

To determine which registers or register fields are double-buffered in this way, see Table 5, "Sensor Register Description," on page 29, the "sync'd-to-frame-start" column.

R0x0D:0[15] can be used to inhibit transfers from the pending to the live registers. This control bit should be used when making many register changes that must take effect simultaneously.

A bad frame is a frame where all rows do not have the same integration time, or where offsets to the pixel values changed during the frame.

Many changes to the sensor register settings can cause a bad frame. For example, when row width R0x03:0 is changed, the new register value does not affect sensor behavior until the next frame start. However, the frame that would be read out at that frame-start has been integrated using the old row width. Consequently, reading it out using the new row width results in a frame with an incorrect integration time.

By default, most bad frames are masked: LINE_VALID and FRAME_VALID are inhibited for these frames so that the vertical blanking time between frames is extended by the frame time.

To determine which register or register field changes can produce a bad frame, see Table 5, "Sensor Register Description," on page 29, the "bad frame" column, and these notations:

- N—No. Changing the register value does not produce a bad frame
- Y—Yes. Changing the register value might produce a bad frame
- YM—Yes; but the bad frame is masked out unless the "show bad frames" feature (R0x0D:0[8]) is enabled

Changes to Integration Time

If the integration time (R0x09:0) is changed while FRAME_VALID is asserted for frame *n*; the first frame output using the new integration time is frame (*n* + *2*). The sequence is as follows:

1. During frame *n*, the new integration time is held in the R0x09:0 pending register.
2. At the start of frame (*n* +*1*), the new integration time is transferred to the R0x09:0 live register.

Integration for each row of frame (*n* + *1*) has been completed using the old integration time. The earliest time that a row can start integrating using the new integration time is immediately after that row has been read for frame (*n* + *1*). The actual time that rows start integrating using the new integration time is dependent on the new value of the integration time.

3. When frame (*n* + *1*) is read out, it is integrated using the new integration time.

If the integration time is changed (R0x09:0 written) on successive frames, each value written is applied to a single frame; the latency between writing a value and it affecting the frame readout remains at two frames.

Changes to Gain Settings

When the gain settings (R0x2B:0, R0x2C:0, R0x2D:0, R0x2E:0, and R0x2F:0) are changed, the gain is usually updated on the next frame start. When the integration time and the gain are changed simultaneously, the gain update is held off by one frame so that the first frame output with the new integration time also has the new gain applied. All the firmware drivers must be off (i.e. seq.cmd=0) before manual control of gains or integration time (R0x09:0) is possible.

Feature Description

PLL Generated Master Clock

The PLL embedded in the sensor core can generate a master clock signal whose frequency is up to 80 MHz (input clock from 6 MHz through 64 MHz). Registers R0x66:0 and R0x67:0 control the frequency of the PLL-generated clock. It is possible to bypass the PLL and use CLKIN as master clock. In order to do so, one must set R0x65:0[15] to 1. If power consumption is a concern, R0x65:0[14] should be also set to 1 a short time later, to put the bypassed PLL in power down mode. To enable the PLL again, the two bits must be set to 0 in the reverse order. By default, the PLL is bypassed and powered down.

PLL Setup

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

R102:0// [15:8] for M; [5:0] for N

R103:0// [6:0] for P

Their relations can be shown by the following equation:

$$f_{PLL}, f_{OUT} = f_{PLL}, f_{IN}, \times M / [2 \times (N+1) \times (P+1)]$$

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

M must be 16 or higher

f_{PFD}, f_{VCO}, f_{OUT} ranges are satisfied

Table 29: Frequency Parameters

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

After determining the proper M, N, and P values and setting them in R102:0/R103:0, the PLL can be enabled by the following sequence:

R101:0[14] = 0// powers on PLL

R101:0[15] = 0// disable PLL bypass (enabling PLL)

Note: If PLL is used, bypass the PLL (R101:0[15]=1) before going into hard standby. It can be enabled again (R101:0[15]=0) once the sensor is out of standby.

PLL Power-up

The PLL takes time to power up. During this time, the behavior of its output clock signal is not guaranteed. The PLL is in the power down mode by default and must be turned on manually. When using the PLL, the correct power-up sequence after chip reset is as follows:

1. Program PLL frequency settings (R0x66:0 and R0x67:0)
2. Power up the PLL (R0x65:0[14] = 0)
3. Wait for a time longer than PLL locking time (> 1ms)
4. Turn off the PLL bypass (R0x65:0[15] = 0)

Window Control

Window Start

The row and column start address of the displayed image can be set by R0x01:0 (Row Start) and R0x02:0 (Column Start).

Window Size

The size of the sensor core image is controlled by R0x03:1 (Row Width) and R0x04:0 (Column Width). The default image size is 1,600 columns and 1,200 rows (UXGA).

The window start and size registers can be used to change the number of columns in the image from 17 through 1,632 and the number of rows from 2 through 1,216. The displayed image size is controlled by the output and crop variables in the mode (ID=7) driver.

Pixel Border

When bits R0x20:0[9:8] are both set, a 4-pixel border is added around the specified image. This border can be used as extra pixels for image processing algorithms. The border is independent of the readout mode, which means that even in skip, zoom, and binning modes, a 4-pixel border is output in the image. When enabled, the row and column widths are 8 pixels larger than the values programmed in R0x03:0 and R0x04:0. If the border is enabled but not shown in the image (R32[9:8] = 01), the horizontal blanking and vertical blanking values are 8 pixels larger than the values programmed in the blanking registers.

Readout Speeds and Power Savings

The sensor core has two ADCs to convert the pixel values to digital data. Because the ADCs run at half the master clock frequency, it is possible to achieve a data rate equal to the master clock frequency. By turning off one of the ADCs, the power consumption of the sensor is reduced. The pixel clock is then reduced by a factor of two.

In R0x20:0 or R0x21:0, bit 10 chooses between the two modes:

- 0: Use both ADCs and read out at the set pixel clock frequency (R0x0A:0[3:0])
- 1: Use 1 ADC and read out at half the set pixel clock frequency (R0x0A:0[3:0])

This can be used, for instance, when the camera is in preview mode. To make the transitions between two sensor settings easier, some simple context switching is described in “Context Switching” on page 129.

Column Mirror Image

By setting R0x20:0[1] = 1 (R0x21:0 in context A), the readout order of the columns are reversed as shown in Figure 20. The starting color is preserved when mirroring the columns.

Row Mirror Image

By setting R0x20:0[0] = 1 (R0x21:0 in context A), the readout order of the rows are reversed as shown in Figure 21. The starting color is preserved when mirroring the rows.

Figure 20: 6 Pixels in Normal and Column Mirror Readout Modes

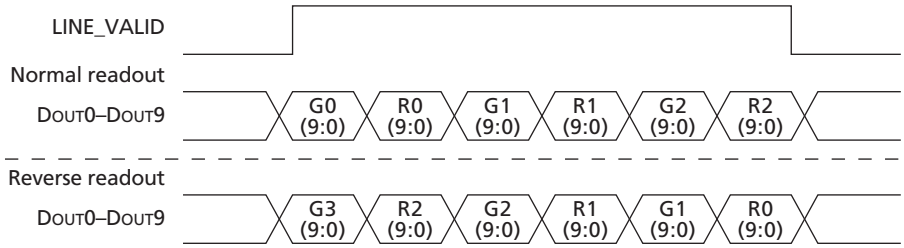
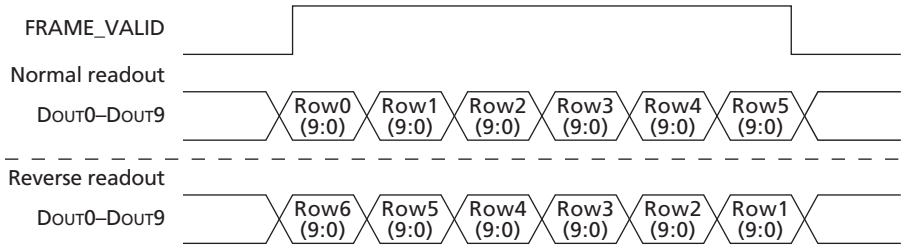


Figure 21: 6 Rows in Normal and Row Mirror Readout Modes



Column and Row Skip

This section assumes context B. If context A is used, replace all references to R0x20:0 with R0x21:0.

By setting R0x20:0[4] = 1 or R0x20:0[7] = 1, skip is enabled for rows or columns, respectively. When skip is enabled, the image is subsampled. The amount of skipping is set by R0x20:0[3:2] (rows) and R0x20:0[6:5] (columns) according to Table 30. Depending on the skip value, the output size variable in the mode (ID=7) driver might need adjustments.

Table 30: Skip Values

Bit Values	Skip Value
00	2
01	4
10	8
11	16

The number of rows or columns read out is what is set in R0x03:0 or R0x04:0, respectively, divided by the skip value in this table.

In all cases, the row and column sequencing ensures that the Bayer pattern is preserved.

Column skip examples are shown in Figures 22 through 26.

Figure 22: 8 Pixels in Normal and Column Skip 2x Readout Modes

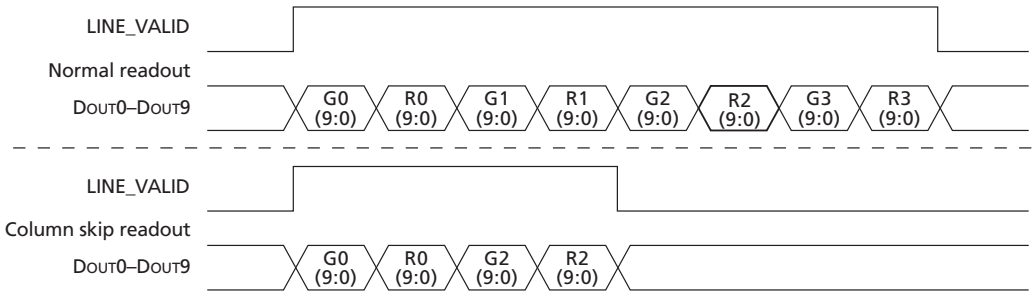


Figure 23: 16 Pixels in Normal and Column Skip 4x Readout Modes

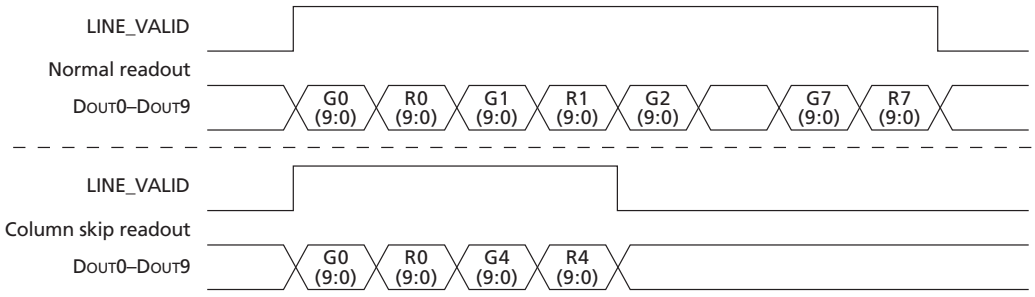


Figure 24: 32 Pixels in Normal and Column Skip 8x Readout Modes

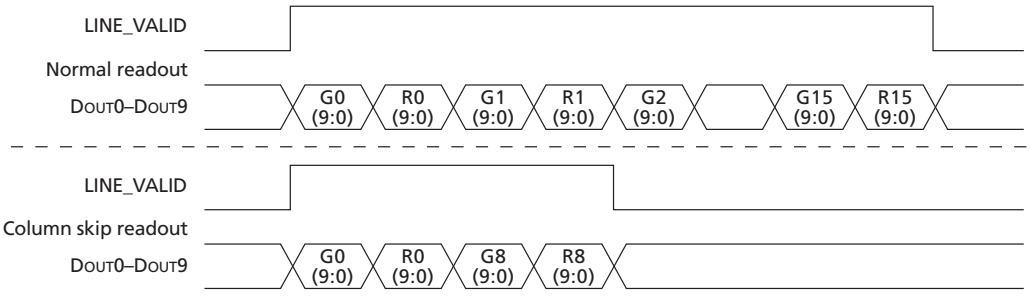
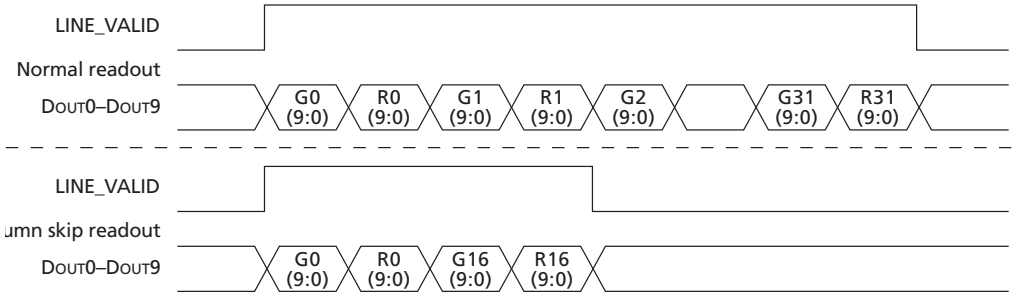


Figure 25: 64 Pixels in Normal and Column Skip 16x Readout Modes



Digital Zoom

Digital zoom is not used in SOC.

Binning

The sensor core supports 2 x 2 binning of 4 pixels of the same color. This mode can be activated by asserting R0x20:0[15] (R0x21:0 if context A is used).

Binning is primarily used instead of 2x skip as a way of decimating the picture without losing information. The effect of aliasing in preview mode is eliminated when binning is used instead of just skipping rows and columns.

Activating binning has a few implications:

- It adds a level of skip, so the picture that comes out has the same dimensions as a picture read out with the next higher skip setting.
- It increases the minimum hblank and minimum row time requirements (see Table 33 and Table 34).

Table 31: Row Addressing

Number of ADCs	Binning	Row Addressing (not considering start address)
1	No	0, 1, 2, 3, 4, 5, 6, 7,...
2	No	0, 1, 2, 3, 4, 5, 6, 7,...
1	Yes	0/2, 1/3, 4/6, 5/7,...
2	Yes	0/2, 1/3, 4/6, 5/7,...

Table 32: Column Addressing

Number of ADCs	Binning	Column Addressing (not considering start address)
1	No	0, 1, 2, 3, 4, 5, 6, 7,...
2	No	0/1, 2/3, 4/5, 6/7,...
1	Yes	0/2, 1/3, 4/6, 5/7,...
2	Yes	0/1/2/3, 4/5/6/7,...

Binning Limitations

To achieve correct operation, the following conditions must be met:

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

Example: Default row size = 1,200. 8x zoom means the actual window on the sensor is divided by 8, so 8x zoom and binning is not allowed with default window size, because 1,200 / 8 = 150, which is not divisible by 4.

- Binning can be seen as an extra level of skip. The combination binning/16x skip is therefore not legal.

Frame Rate Control

For a given window size, the blanking registers (R0x05:0 - R0x08:0) along with the row speed register (R0x0A:0) can be used to set a particular frame rate.

The frame timing equations (Table 27 and Table 28 on page 120) can be rearranged to express the horizontal blanking or vertical blanking values as a function of the frame rate:

HBLANK_REG = master clock freq / (frame rate*
((R0x03:0/S + BORDER) + VBLANK_REG)*PIXCLK_PERIOD) - (R0x04:0/S + BORDER)

VBLANK_REG = master clock freq / (frame rate*
((R0x04:0/S + BORDER) + HBLANK_REG)*PIXCLK_PERIOD) - (R0x03:0/S + BORDER)

The HBLANK_REG value allows the frame rate to be adjusted with a minimum resolution of one PIXCLK_PERIOD multiplied by the total number of rows (displayed plus blanking). When finer resolution is required, R0x0B:0 (extra delay) can be used. R0x0B:0 allows the frame time to be changed in increments of pixel clocks.

iiX4X4ontal Blanking

The minimum horizontal blanking value is constrained by the time used for sampling a row of pixels and the overhead in the row readout. This is expressed in Table 33.

Table 33: Minimum Horizontal Blanking Parameters

Parameter	Default / 2 ADC Mode, No Binning	1 ADC Mode, No Binning	2 ADC Mode, Binning	1 ADC Mode, Binning
HBLANK(MIN)	286 mclks	324 mclks = 162 pixclks	470 mclks	508 mclks = 254 pixclks

Minimum Row Time Requirement

The total row time must be sufficient to allow all row operations (readout and shutter operations). The row time is the sum of column width (halved during binning divided by column skip factor) and horizontal blanking, and can therefore be adjusted by programming these.

Table 34 shows minimum row time as a function of mode of operation.

This is a particularly strict requirement during binning because twice as many row operations are required per row and the column width is halved.

Table 34: Minimum Ro

Parameter	Default / 2 ADC Mode, No Binning	1 ADC Mode, No Binning	2 ADC Mode, Binning	1 ADC Mode, Binning
ROW_TIME(MIN)	473 mclks	488 mclks = 244 pixclks	931 mclks	946 mclks = 473 pixclks
pointer_operations	461 mclks	464 mclks	919 mclks	922 mclks

Context Switching

When the sensor is in the SOC bypass mode, R0xF2:0 is designed to enable easy switching between sensor modes. Some key registers and bits in the sensor have two physical register locations, called contexts. Bits 0, 1, and 3 of R0xF2:0 control which context register context is currently in use. A "1" in a bit selects context B, while a "0" selects context A for this parameter. The select bits can be used in any combination, but by default are setup to allow easy switching between preview mode and full resolution mode:

Context B (Default context)

R0xF2:0	= 0x000B	(Context B)
R0x05:0	= 0x015C	(Horizontal blanking, context B)
R0x06:0	= 0x0020	(Vertical Blanking, context B)
R0x20:0	= 0x0000	(2 ADCs, no column or row skip)

Description: Full resolution UXGA (1,600 x 1,200) image at 15 fps

Context A (Alternate context, preview mode)

R0xF2:0	= 0x0000	(Context A)
R0x07:0	= 0x00AE	(Horizontal blanking, context A)
R0x08:0	= 0x0010	(Vertical blanking, context A)
R0x21:0	= 0x0490	(1 ADC, 2x column and row skip)

Description: Half-resolution SVGA (800 x 600) image at 30 fps

The horizontal blanking and vertical blanking values for the two contexts are chosen so that row time is preserved between contexts. This ensures that changing contexts does not affect integration time. A few more control bits are also available through the context register (R0xF2:0) so that flash and restarting the sensor can be done simultaneously with changing contexts. See Table 5, "Sensor Register Description," on page 29 for more information.

Settings for skip, 1 ADC mode, and binning can be set separately for context B and context A using R0x20:0 and R0x21:0, respectively. When these settings are referred to in this document, the register is dependent on what context is set in R0xF2:0. For the default (no bypass) mode, context switching is controlled by the sequencer (ID=1) driver.

Integration Time

Integration time is controlled by R0x09:0 (shutter width in multiples of the row time) and R0x0C:0 (shutter delay, in PIXCLK_PERIOD/2). R0x0C:0 is used to control sub-row integration times and only has a visible effect for small values of R0x09:0. The total integration time, ^{t}INT, is shown in the equation below:

^{t}INT = R0x09:0 * Row Time - Integration Overhead - Shutter Delay

where:

Row Time = (R0x04:0/S + BORDER + HBLANK_REG)*PIXCLK_PERIOD master clock periods (from Table on page 120)

S = Skip Factor, multiplied by 2 if binning is enabled

Overhead Time = 260 master clock periods (262 in 1 ADC mode)

Shutter Delay = R0x0C:0 * PIXCLK_PERIOD master clock periods (/2 in 1 ADC mode)

with default settings:

^{t}INT = (1,232 * (1,600 + 348)) - 260 - 0
= 2,399,676 master clock periods = 66.66ms at 36 MHz

In the equation, the integration overhead corresponds to the delay between the row reset sequence and the row sample (read) sequence.

Typically, the value of R0x09:0 is limited to the number of rows per frame (which includes vertical blanking rows), so that the frame rate is not affected by the integration time. If R0x09:0 is increased beyond the total number of rows per frame, the sensor adds blanking rows as needed. Additionally, ^{t}INT must be adjusted to avoid banding in the image caused by light flicker. Therefore, ^{t}INT must be a multiple of 1/120 of a second under 60Hz flicker, and a multiple of 1/100 of a second under 50Hz flicker.

Maximum Shutter Delay

The shutter delay can be used to reduce the integration time. A programmed value of *N* reduces the integration time by *N* master clock periods. The maximum shutter delay is set by the row time and the sample time, as shown in the equation below:

Maximum shutter delay = (Row Time - pointer_operations)

where:

Row Time = (R0x04:0/S + BORDER + HBLANK_REG)*PIXCLK_PERIOD master clock periods (from Table on page 120)

S = Skip Factor, multiplied by 2 if binning is enabled

pointer_operations = see Table 34 on page 129.

with default settings:

Maximum shutter delay = (1,600 + 348) - 461

= 1,487 (master clock periods)

If the value in this register exceeds the maximum value given by this equation, the sensor may not generate an image.

Flash STROBE

The sensor core supports both Xenon and LED flash through FLASH output. The timing of FLASH with the default settings is shown in Figure 26, Figure 27, and Figure 28. R0x23:0 allows the timing of the flash to be changed.

The flash can be programmed to:

- fire only once
- be delayed by a few frames when asserted
- and (for Xenon flash) the flash duration can be programmed

When Xenon flash is enabled, an integration time significantly smaller than one frame causes uneven exposure of the image, as does setting a flash pulse width larger than vertical blanking.

Enabling the LED flash causes one bad frame in which several rows have the flash on during only part of their integration time. This can be avoided by forcing a restart (write R0x0D:0[1] = 1) immediately after enabling the flash; the first bad frame is then masked out as shown in Figure 28. Read-only bit R0x23:0[14] is set during frames that are correctly integrated; the state of this bit is shown below.

Figure 26: Xenon Flash Enabled

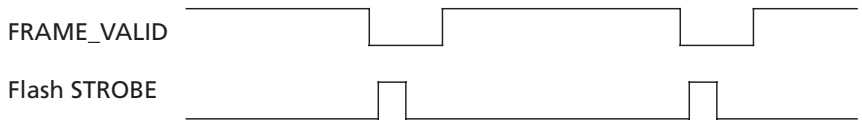
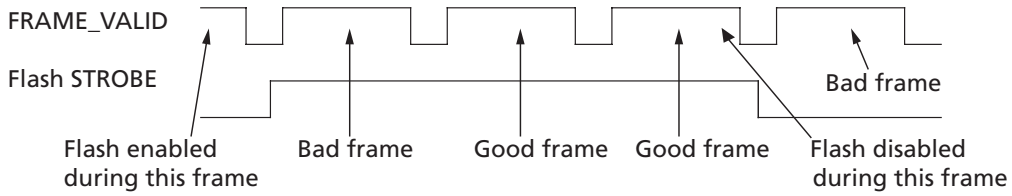
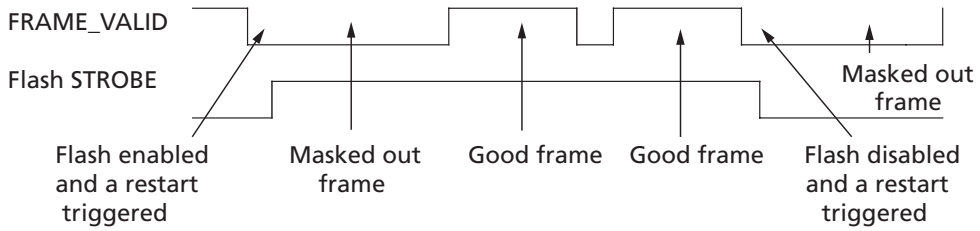


Figure 27: LED Flash Enabled



Note: Integration time = number of rows in a frame.

Figure 28: LED Flash Enabled, Using Restart



Note: Integration time = number of rows in a frame.

Global Reset

The sensor core provides a global reset mode in which the pixel integration time is controlled by an external mechanical shutter. The sensor can then operate on a lower clock frequency, reducing the bandwidth on the interface between the sensor and the host processor without losing image quality.

The basic operation is as follows:

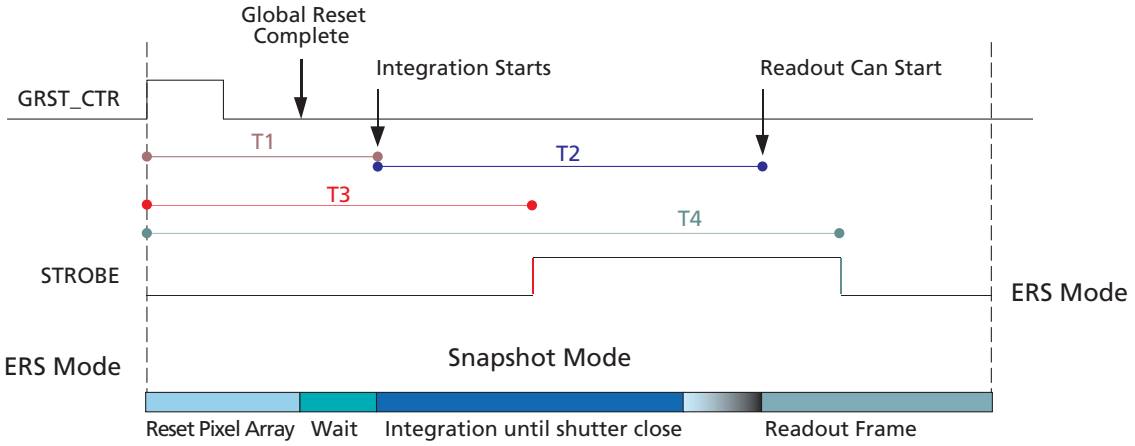
1. The sensor operates in either preview or full-frame mode (electronic rolling shutter [ERS]).
2. A rising edge on the signal GRST_CTR or a WRITE to an internal register starts the global reset sequence.
3. The sensor now enters the snapshot mode and after a certain time, all the lines in the sensor array are reset and kept in a reset state until the integration starts.

The start of the integration (exposure) period, the assertion of STROBE, the start of the readout, and the de-assertion of STROBE can be controlled by internal registers (T1, T2, T3, and T4, as shown in Figure 29).

The sensor core provides an output signal, STROBE, that can be used to control the mechanical shutter. This signal can be programmed to occur in a specified window around the actual start of integration. During global reset, FLASH is programmed in a different way than during normal ERS operation. Normally, the FLASH behavior is programmed using R0x23:0. In global reset mode, FLASH is programmed in the same way as STROBE, showed in Figure 29, using registers R0xC5:0 and R0xC6:0.

R0xC0:0[0] controls the mechanism for starting the readout after a GLOBAL RESET operation. If this bit is HIGH, the integration time is directly controlled by GRST_CTR. Very long integration times can be achieved this way.

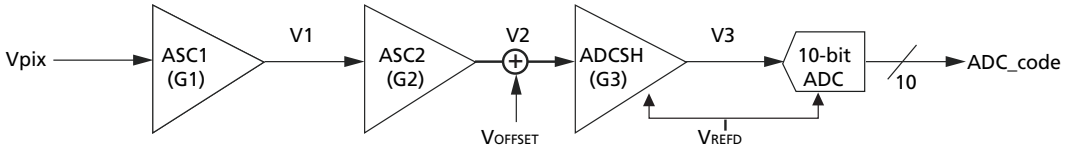
Figure 29: GLOB



Analog Signal Path

The sensor core features two identical analog readout channels. A block diagram for one channel is shown in Figure 30. The readout channel consists of two gain stages (ASC1 and ASC2), a sample-and-hold (ADCSH) stage with black level calibration capability (V_{OFFSET}), and a 10-bit ADC.

Figure 30: Analog Readout Channel



Stage-by-Stage Transfer Functions

Transfer functions proceed stage-by-stage, as follows:

Let VPIX be the input of the signal path:	VPIX = pixel output voltage = signal path input voltage,	
The output voltage of ASC 1st stage is:	V1 = -1*G1*VPIX	(1)
The output voltage of ASC 2nd stage is:	V2 = -1*G2*V1	(2)
The output voltage of ADC Sample-and-Hold stage is:	V3 = 2*G3*V2 - VREFD + VOFFSET	(3)
and the ADC output code is:	ADC output code = 511*(1 + (V3 / VREFD))	(4)
From (1) to (4), the ADC output code can also be written as:	ADC code = (1022/VREFD)*[G1*G2*G3*VPIX + (Voffset/(2*G3))]	(5)

Where G1, G2, and G3 are the gain settings, VOFFSET is the offset (calibration) voltage, and VREFD is the reference voltage of the ADC. The gain setting G3 is applied to the signal but is not applied to VOFFSET. The parameters VREFD, G1, G2, G3, and VOFFSET are described next.

VREFD

The VREFD parameters are as follows:

The ADC reference voltage VREFD is:	VREFD = VREF_HI - VREF_LO	(6)
where	VREF_HI = 55.5mV*(R0x41:0[7:4] + 23)	(7)
using default register values:	VREF_HI = 55.5mV*(13 + 23) = 1.998V	
and	VREF_LO = 55.5mV*(R0x41:0[3:0] +11)	(8)
using default register values:	VREF_LO = 55.5mV*(7 +11) = 0.999V	
so	VREFD = 55.5mV*(R0x41:0[7:4] - R0x41:0[3:0] + 12)	(9)
using default register values	VREFD = 1.998 - 0.999 = 0.999V	

Gain Settings: G1, G2, G3

The gains for green1, blue, red, and green2 pixels are set by registers R0x2B:0, R0x2C:0, R0x2D:0, and R0x2E:0, respectively. Gain can also be globally set by R0x2F:0. The analog gain is set by bits 8:0 of the corresponding register as follows:

G1 = bit 7 + 1	(10)
G2 = bit 6:0 / 32	(11)
G3 = bit 8 + 1	(12)

Digital gain is set by bits 11:9 of the same registers.

Offset Voltage: VOFFSET

The offset voltage provides a constant offset to the ADC to fully utilize the ADC input dynamic range. The offset voltages for green1, blue, red, and green2 pixels are manually set by registers R0x61:0, R0x62:0, R0x63:0, and R0x64:0, respectively. The offset voltages also can be automatically set by the black level calibration loop.

For a given color, the offset voltage, VOFFSET, is determined by:

$$V_{OFFSET} = 0.50V * offset\_gain * offset\_sign * offset\_code[7:0]/255 \quad (13)$$

where:
"offset_sign" is determined by bit 8 as:
if bit 8 = 0, offset_sign = +1 (14)
if bit 8 = 1, offset_sign = -1 (15)
"offset_code" is the decimal value of bit<7:0>

OFFSET_GAIN is determined by the 2-bit code from R0x5A:0[1:0], as shown in Table 35. These step sizes are not exact; increasing the stage0 ADC gain from 2 to 4 decreases the step size significance; decreasing the ADC VREFD increases the step size significance.

Table 35: Offset Gain

R0x5A:0[1:0]	OFFSET_GAIN
00	OFFSET_GAIN = 0 (no calibration voltage is applied)
01	OFFSET_GAIN = 0.25 (1 calibration LSB is equal to 0.5 ADC LSB when VREFD = 1V)
10	OFFSET_GAIN = 0.50 (1 calibration LSB is equal to 1 ADC LSB when VREFD = 1V)
11	OFFSET_GAIN = 1 (1 calibration LSB is equal to 2 ADC LSB when VREFD = 1V)

Recommended Gain Settings

The analog gain circuitry in the sensor core provides signal gains from 1 through 15.875.

Table 36: Recommended Gain Settings

Desired Gain	Recommended Gain Register Setting
1–1.969	0x020–0x03F
2–7.938	0x0A0–0x0FF
8–15.875	0x1C0–0x1FF

Analog Inputs AIN1–AIN3

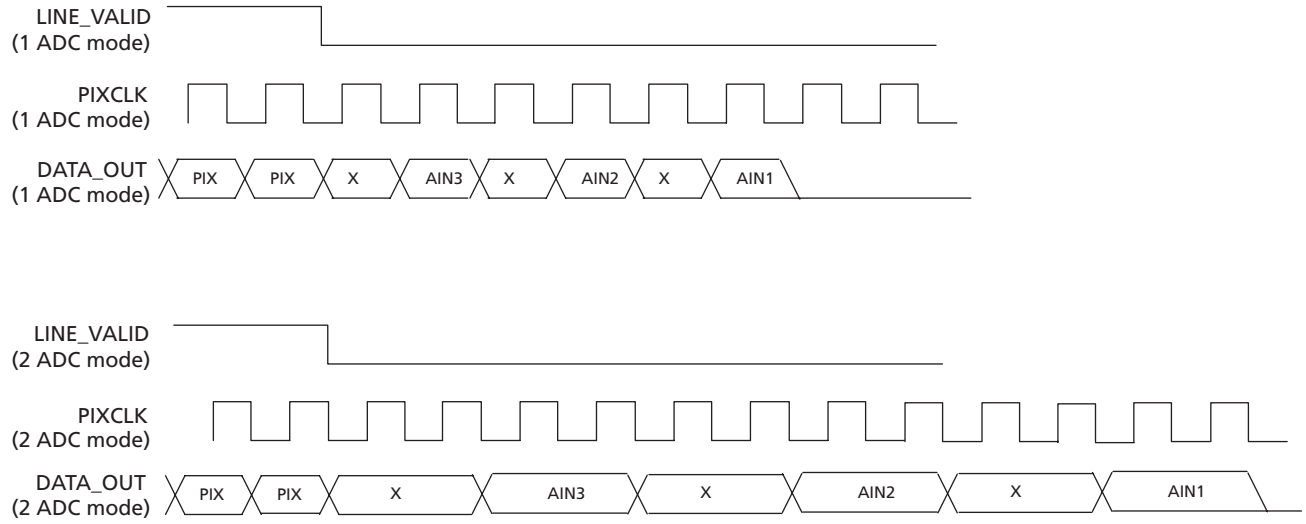
Since ADC resources in the sensor core are not used to digitize pixel array output 100 percent of the time, they can be intermittently used to sample external analog signals coming from a variety of sources—e.g., a flash charging circuit or an auto focus lens actuator. If R0xE3:0[15] = 1, the chip samples AIN1–AIN3 once per every pixel array row (after reading out the row). Digital data produced by this sampling are available to the user 8w 84

- They can be read from registers R0xE0:0 through R0xE2:0.
- If R0xE3:0[14] = 1, the data are additionally inserted into image data stream each time LINE_VALID goes LOW.

The nominal range of AIN are 0V + VOFFSET to VREFD + VOFFSET. VREFD is the ADC reference voltage (nominally 1V), but can be programmed. (See "Analog Signal Path" on page 132.) VOFFSET is the offset in the ADC and is typically ±10mV to 20mV. If required, the offset can be measured by converting a calibrated reference voltage, which can be

used to compensate at the input. The ADC is designed to operate with differential inputs. Since AIN1–AIN3 are used as single-ended inputs to the ADC, it is recommended to average values from several samples (if possible, a whole frame) to cancel out noise.

Figure 31: Timing Diagram AIN1–AIN3 Sample



Firmware

Firmware implements all automatic functionality of the camera, including auto exposure, white balance, auto focus, flicker detection and so on. The firmware consists of drivers, generally one driver per each major control function. The firmware runs on a 6811-compatible microcontroller with a mathematical co-processor. 32K of metal mask-programmable PROM is available for non-volatile code. 2K of volatile SRAM is available, where typically the first 1K of SRAM is used by the PROM drivers for data and stack, while to the second 1K is reserved for loadable custom code.

The firmware and MCU subsystem targets to achieve the following high-level goals and features:

- execute applications from ROM and RAM
- execute applications in real-time, synchronized to IFP
- allow loading custom executable programs
- firmware controls IFP by reading and writing ring bus registers
- firmware consists from drivers and bootstrap code
- user can extend or override functionality of ROM drivers
- user can add all new drivers
- drivers have variables; user configures drivers by setting variables via two-wire serial interface
- when idle, puts MCU into low-power sleep mode
- a minimal code overhead due to flexibility
- transparency from firmware build version
- stability, diagnostics and recovery from software crashes
- framework for application and driver development

The firmware is written in C with some inline assembler where extra performance is required. It runs on 6811-compatible microcontroller with a mathematical co-processor.

Overview of Architecture

Bootstrap Routine

The firmware consists of a bootstrap code, drivers and a test module. The bootstrap code initializes internal low-level MCU registers, performs a mini-self-test, and sets up a 128-bytes deep stack. The main routine is then invoked.

Drivers

Firmware applications are organized into drivers. Each driver performs a specific function. Each driver has its own data structure with various variables. The user can access the data structure via R198:1 and R200:1 to read or write variables. The access is implemented using hardware direct memory access (DMA).

Drivers are static objects with virtual methods. Each driver has an associated data structure and a driver ID. To access a driver variable, the user must know driver ID and variable offset in the data structure. Driver methods are implemented using a virtual method table (VMT). VMT is a table of pointers to driver methods. The object data structure has a pointer to VMT. Software routines must call methods indirectly, via VMT pointer. This allows overriding of methods.

Extending and Overriding Drivers

VMTs are necessary to be able to override a driver. To override a driver, the software developer has to perform the following steps:

1. Load executable code

2. Construct new VMT
   a. use pointers to the new functions
   b. use old pointers to call old methods
3. New methods can call inherited functions using old VMT pointers
4. Point driver data structure VMT pointer to the new VMT

The main routine continues firmware initialization by calling driver initialization functions. If all drivers initialization completes without error, the main function indicates a successful firmware start by setting a code in R195:1. The user can request the firmware to start in a special test mode. The bootstrap code reads R195:1 and invoke the corresponding routine.

Variables

The main routine sets up a system table of pointers to drivers. The address of the table is recorded in SFR 0x1050. The hardware DMA uses this register to implement logical access to variables.

MCU SRAM can be accessed in two ways:

1. By specifying a physical address (physical mode)
2. By specifying driver ID and variable offset in driver's data structure (logical mode)

In physical mode, the user writes address to be accessed (read or written) in R198:1 and reads or writes data in R200:1. PROM data is not accessible using this mechanism.

In logical mode, the user writes driver ID and variable offset combination in R198:1 and reads or writes data in R200:1. This mode allows accessing driver variables regardless of their physical location, which can change from one firmware version build to another. In this sense variables are very much like hardware registers.

Uploading Custom Code

To upload custom code

1. Write application data to SRAM using physical variable access
2. Use monitor driver to invoke the code

See “Monitor Driver” on page 139 for details.

Hardware Resource Programming

The MCU subsystem provides a number of hardware resources to the software developer. These resources include

DMA To enable user access variables.

Watchdog The watchdog monitors execution of firmware and resets MCU when a time out is detected. During normal execution the firmware must reset the watchdog periodically or disable it.

Sleep and wakeup Logic is available to suspend MCU clock and enter low-power mode. The MCU can wake up synchronously with IFP when IFP is processing a line with a specified number.

Internal Register Bus Access

The firmware has master access to the ring bus and can read and write any hardware register.

GPIO and Waveform Generator on Local Bus

For better performance GPIO and waveform generator are connected to the local 6811 SFR bus.

High-Precision Timer A 32-bit master clock (80 MHz) counter for code profiling or time measurements.

Sleep and Wakeup Programming

To sleep and wake up on a certain line, program the following:

1. Set line number to wake up at SFR 0x1042.
   a. User can choose to wake up on IFP line, referred to interpolation, or frame enable of IFP, sensor, JPEG, and FIFO. The edge and level are selectable.
2. Clear SFR 0x1040[0].
   b. This clears previous sleep event notification.
3. Read from SFR 0x1040 to attempt sleeping.
   c. MCU tries entering sleep mode. MCU does not sleep if DMA is happening or the wakeup line number is reached or GPIO notification signal is asserted.
4. Read SFR 0x1040 to check if sleep succeeded

Note: If bit 0 is set, sleeping was successful. Otherwise repeat step 3.

A library function is available for easy sleep mode usage.

Two-Wire Serial Interface Programming

The microcontroller can access all registers on the ring bus. The access is enabled by custom SFRs located at addresses 0x1060 through 0x1066. To carry out a bus read, set desired I/O page in READ_IOPAGE and write desired register address to READ_IOADR to initiate the READ transaction. Poll IOSTATUS until it returns a “0.” The read data is available in IODATA.

To carry out a bus write, set the desired I/O page in WRITE_IOPAGE. and the register address in WRITE_IOADR. Load write data into IODATA to initiate a WRITE transaction. Poll IOSTATUS until it returns a “0” to signal the completion of the WRITE transaction.

A library function is available for easy sleep usage.

DMA Programming

Set up table of variables for logical access. Driver ID corresponds to the pointer offset in the table. Program table address into SFR 0x1050.

Warm Restarts and Watchdog Programming

The watchdog register can detect a software crash and have the system recover from it by issuing a hardware reset. The watchdog resets the microcontroller subsystem if SFR 0x1040 register have not been cleared for *N* frames.

Number *N* is programmed in the WATCHDOG_REG. Setting *N* to “0” disables the watchdog. The user may need this option when the microcontroller runs a lengthy task, e.g., calculation of CRC or RAM testing.

When the watchdog resets the microcontroller, OS restarts and checks for restart reason. Register RESTART_CODE specifies the probable code. The user can also see restart code by reading R195:1. A restart caused by resetting the chip is called a "cold" restart. Restart caused by the watchdog, software, or illegal code trap are called "warm" restarts.

High-Precision Timer

SFR 0x1048 is a high-precision timer is available for use in applications, and profiling code during development. It is a 32-bit counter incremented every master clock, except when in standby. To profile, read and store the timer before executing a routine. After the routine exits, read the timer again and subtract from the initial value.

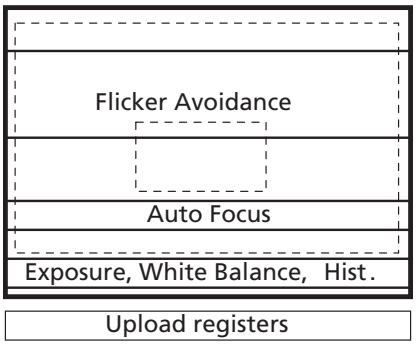
Safe and Test Modes

A safe mode is available to start MCU without running drivers. Set R195:1 and reset MCU to enable this mode. Only monitor runs allowing to start ROM or load and start RAM code.

R195:1 can also invoke the MCU and memory test mode.

Application Scheduling

Drivers run in synchrony with IFP image processing. As the frame comes through the color pipeline in real time, certain statistics are calculated and become available. Typically, the flicker avoidance driver is active at the top of the frame. Auto focus measurement window is located in the center of the image. Once its result is ready, the auto focus and auto focus mechanics drivers execute. The exposure, white balance, and histogram statistics windows nearly cover the entire image. Their result is available at the end of the frame, when AE, WB, and histogram drivers are invoked. Finally, during the vertical blanking, drivers upload calculated settings to sensor and SOC.

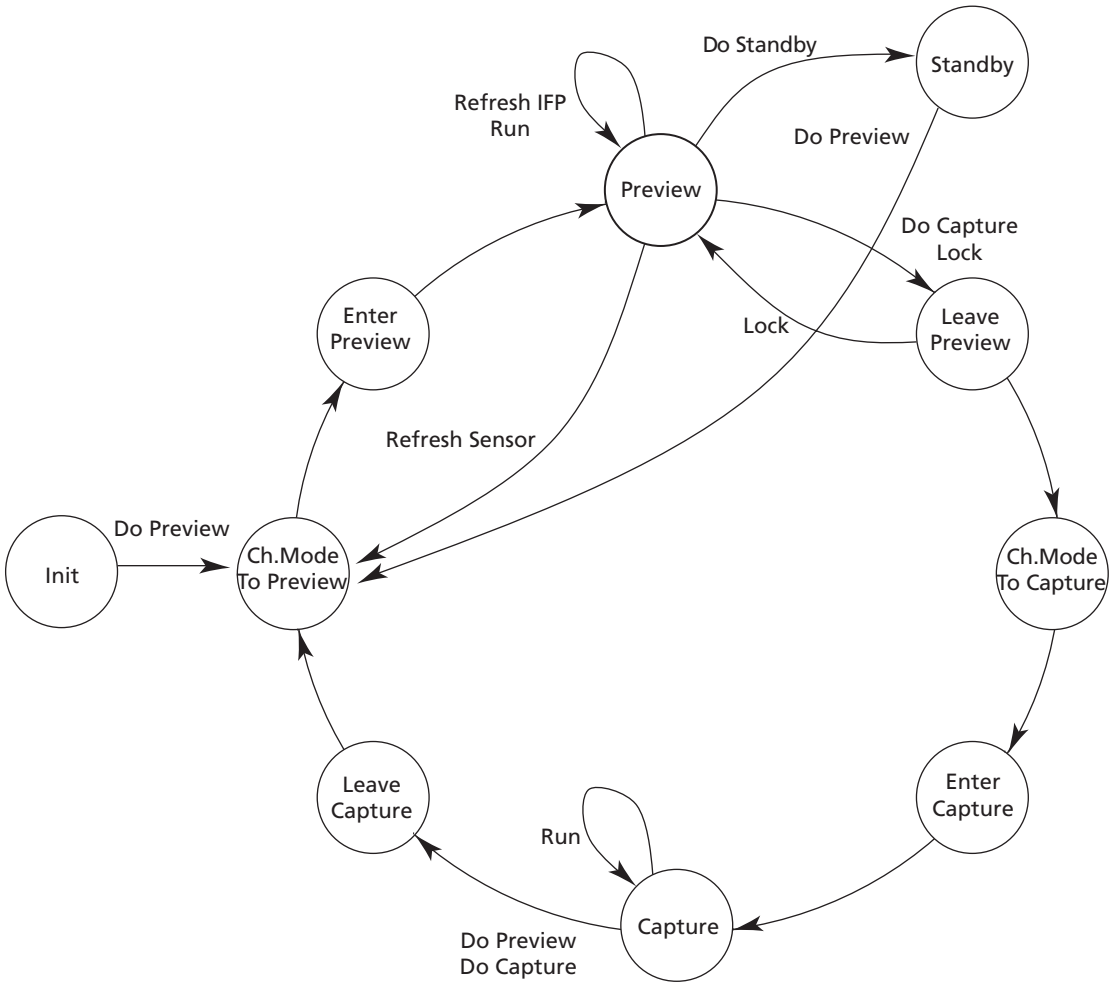


Monitor Driver

The main monitor function is to run code. Set the "arg1" to the address of the function to be called, set “arg2” to an optional 16-bit parameter to be passed and set "cmd" to 0x01. The sequencer driver calls the monitor driver typically once per frame. The desired function is invoked in one frame time or less. The monitor also provides CRC calculation capability to check if the uploaded firmware was transmitted correctly.

Variable mon.ver contains firmware version number. mon.ver[7] = 1 indicates uploaded firmware.

Figure 32: Sequencer Driver



Sequencer Driver

The sequencer is a finite state machine that controls the operation of main functions and the switching between modes. Camera operation is organized as states, such as preview or capture. The sequencer carries out a number of programs, such as previewing, preview lock, capture, and so on.

The state of the sequencer is indicated in variable state. To have the sequencer execute a certain program, set the program number in cmd. The host then should monitor state to know when to change resolution and/or capture frames.

Each state has its configuration (see "Firmware Driver Variables" on page 68). Before executing programs, set up state configurations to customize the program. For example, to capture compressed frames, enable compression in capture state configuration.

A typical scenario includes the following:

1. Configure mode variables after hardware reset
    a. Set up sensor image size for preview
    b. Set up displayed image size
    c. Set up FIFO to smooth data rate
2. Configure preview mode

a. Set auto exposure, white balance and auto focus speed

3. Execute sensor REFRESH command
4. Run in preview until shutter button is depressed
   a. If the shutter button is depressed halfway, execute the lock program
      i. Configure preview state to disable necessary functions to be locked, e.g., auto exposure, white balance, and auto focus
      ii. Configure PreviewLeave state for fast settling of those functions
      iii. Execute the lock program
      iv. The lock program transits PreviewLeave state, perform fast settling, and return to Preview state, which was configured to freeze (lock)
   b. If the shutter button is depressed all the way, execute the capture program
      i. If already in lock mode, disable PreviewLeave state and execute capture program
      ii. If not in lock state and some settling is required (auto focus), configure PreviewLeave state for fast settling
      iii. Execute capture program. The program transitions to PreviewLeave, perform required settling and proceed through mode change to capture
5. Capture frame
   a. Preconfigure capture mode variables for correct image size, compression, and select video/still
   b. Monitor the "state" variable. When the state is capture, grab the frame(s)

Optionally, configure Capture Enter or Preview Leave to have output blanked. This helps grabbing correct frame for capture.

Flash Types and Settings

The sequencer supports three types of flash light source:

- LED
- Xenon
- Xenon continuous

Flash type is selected using seq.sharedParams.flashType.

States in the sequencer diagram have an associated flash setting. That setting is specified using variables such as seq.prevParEnter.flash.

The four settings available are shown in Table 37.

Table 37: Flash Types and Settings

Type	Setting
MODE_FLASH_OFF	Turns flash off
MODE_FLASH_ON	Turns flash on
MODE_FLASH_AUTO	Checks for low-light condition and turns flash on if necessary
MODE_FLASH_LOCKED	Turns flash on or off to put it into the state selected by MODE_FLASH_AUTO

Using LED Flash

To perform CM_LOCK

1. Set PreviewLeave flash configuration to MODE_FLASH_AUTO. Set the state's AE and WB to fast settling
2. Set Preview flash configuration to MODE_FLASH_OFF
3. Perform CM_LOCK. That turns flash on in low-light conditions, perform fast AE/WB settling, return to Preview and turn flash off

To continue with capture after CM_LOCK

1. Disable all automatic functions in PreviewLeave. Set flash mode to MODE_FLASH_LOCKED
2. Disable CaptureEnter state
3. Perform CM_CAPTURE. That restores flash state in PreviewLeave and make a snapshot

To perform CM_CAPTURE without CM_LOCK

1. Set PreviewLeave flash configuration to MODE_FLASH_AUTO. Set the state's AE and WB to fast settling.
2. Disable CaptureEnter state
3. Perform CM_CAPTURE. That automatically turns flash on, perform fast settling and make a snapshot

Note: When the LED flash is turned on, the current frame coming out has not seen the effect of the LED; therefore, use the SkipFrame variable to skip a frame when turning the LED on.

Using Xenon Burst Flash

This type of flash fires on every frame. In many respects it is similar to LED, except that the sensor integration time must always be more than 1 frame.

Using Xenon Flash

The three flash types have their own specifics.

To capture images using one-shot Xenon, do the following.

1. If using CM_LOCK, carry out the lock operation as usual
2. Before issuing CM_CAPTURE, re-program the sequencer to disable automatic advancement from state to state, seq.StepMode
3. Reprogram CaptureEnter state to have all automatic functions off (AE, AF, WB, FD)
4. Advance to ModeChange and load Xenon-specific parameters directly into the color pipeline

- color matrix for Xenon R96-R102:1
- integration time R9:0 (1 full frame or more)
- analog gains, R0x2B–0x2E:0
- digital gains R78:1, R106–109:1

Low-Light Operation

A number features are available to improve image quality in low-light conditions:

- increased noise suppression in interpolation
- reduced color saturation
- reduced aperture correction
- increased aperture correction threshold
- Y-filter

The seq.sharedParams.LLmode variable enables individual low-light features. seq.sharedParams.LLvirtGain1 and 2 specify gain thresholds to enable some of these features. Values are given by seq.sharedParams.LLInterpThresh1/2, LLSat1/2, LLApCorr1/2 and LLApThresh1/22.

Firmware

Auto Exposure Driver

The AE driver works to achieve desirable exposure by adjusting sensor core's integration time, analog, and digital gains. The driver can be configured with respect to desired AE speed, maximum and minimum frame rate, the range of gains, brightness, backlight compensation, and so on.

AE driver typically runs in one of these modes. The modes are set in the sequencer driver for each sequencer state.

- fast settling preview—reach target exposure as fast as possible
- continuous preview—a slow-changing mode good for video capture
- evaluative—evaluate current scene and adjust exposure for still capture

Some key variables affecting all modes:

- ae.Target— controls target exposure for all modes. Increasing or decreasing this variable makes the image brighter or darker
- ae.Gate—how accurate AE driver tracks the target exposure
- ae.weights—specify weights for central and peripheral backlight compensation in preview modes

AE driver adjusts exposure by programming sensor gains, R43–46:0 and R65:0, sensor integration time R9:0 and R12:0 and IFP digital gains R78:2 and R110:1.

Evaluative Auto Exposure

Evaluative AE (EAE) selects optimum exposure for scenes where conventional AE does not give good results:

- scenes involving sun
- back light scenes
- strong contrast scenes and so on

EAE breaks down input image into a 4 x 4 grid of subwindows and analyzes their exposure value (EV) readings. It evaluates brightness and contrast in the image, the scene and adjusts exposure appropriately. Variables ae.mmEVZone1/2/3/4 keep programmable thresholds defining brightness classes. Variable ae.mmShiftEV is used to calibrate EV readings for particular module type.

Auto White Balance Driver

AWB detects the temperature of the light source in the scene and adjusts color correction to always produce image for sRGB display. In other words, it makes the gray areas in the raw image look also gray in the output image. To detect the illuminant temperature, the driver reads results generated by the statistics block in the color pipeline, R48–50:1. Color correction is achieved via adjusting sensor analog RGB gains R43–46:0, IFP digital RGB gains R106–109:1, and coefficients of the color correction matrix R96–102:1.

The adjustment is done in two ways:

The driver adjusts RGB gains to achieve a gray output. When gains are too large, the driver also adjusts the CCM. During part calibration, two sets of CCM coefficients are obtained, one for red-rich, incandescent illumination and one for blue-rich daylight illumination. These two sets are specified in the AWB driver using awb.ccmL and awb.ccmRL arrays of variables. Variable awb.ccmL corresponds to the red-rich set, also called left. Variable awb.ccmRL is the delta between blue-rich—or right—and the left sets. CCM programmed into the color pipeline is available in variables awb.ccm. The current setting is calculated by interpolating between the left and right sets. The awb.CCM position indicates the position, from 0–left, red-rich to 127–right, blue-rich.

For example, cool-white office illumination tends to have a position near 64. Variables awb.CCMpositionMin, awb.CCMpositionMax, awb.GainMin, awb.GainMax specify CCM position and digital RGB gain limits. When changing these settings, issue a REFRESH command to put new values into effect.

AWB speed is controlled by awb.JumpDivisor and awb.GainBufferSpeed. AWB statistics window typically covers the entire image. Its size is set by awb.windowPos and awb.windowSize. Issue a REFRESH_MODE command for settings to take effect. AWB can be forced to daylight when AE detects outdoor condition, see seq.mode[5].

AWB can be operated manually in two ways:

1. Set awb.mode[5] = 1. This forces digital RGB gains to unity. Program desired CCM position into awb.CCMposition.
2. Set awb.CCMpositionMin and awb.CCMpositionMax limits to desired position. Issue REFRESH command. CCM moves into specified position. To fix gains to unity, set awb.GainMin and awb.GainMax to 128. Otherwise digital gains continue changing automatically.

Auto Focus Driver

The AF driver works to find best lens position providing maximal image sharpness. The driver operates in two modes: manual and auto mode. In manual mode (af.mode[7] = 1) focus position is defined by variable af.bestPosition. User can change lens position from af.posMin to af.posMax.

Automatic mode is started by af.mode=1, The driver automatically scans af.numSteps focal zones to find best position. AF driver makes search of the best focus position based on average sharpness and luma information received from 16 programmable windows. Each filter kernel has seven user-programmable elements specified in registers R75:2, R75:2, R85:2 and R86:2. User can specify window size and position (af.windowPos, af.windowSize), number of the focal zones to scan (af.numSteps), weight for each window (af.zoneWeights).

Public variables of the AF driver listed in Table 15, "Driver Variables-Auto Focus Driver (ID = 5)," on page 82 and register settings defining the AF filters (see page 14) are all the parameters that one needs to pay attention to when customizing the built-in AF algorithm. The AF driver variables include two unsigned characters (bytes) named af.windowPos and af.windowSize that should be used to adjust position and size of the AF windows instead of direct writes to registers R[64:66]:2. It is certainly possible to access these registers directly, and new values written to them have immediate effect. However, these values remain in effect only as long as Sequencer driver, the master firmware driver continuously running on MT9D111 microcontroller unit (MCU), does not call AF driver function AF_SetSize (or its user-supplied substitute). The Sequencer calls this function at its initialization, at every change of sensor operation mode (e.g. from preview mode to capture mode), and also whenever sq.cmd variable is set to 6 in the preview mode. The function AF_SetSize translates the current settings of af.windowPos and af.windowSize to corresponding settings of registers R[64:66]:2 and writes these settings over the previous values of the registers. There is no way to change the precedence of af.windowPos and af.windowSize over the register settings other than by overriding the function AF_SetSize.

In addition to programming the registers R[64:66]:2, the function AF_SetSize automatically sets af.wakeUpLine in accordance with af.windowPos and af.windowSize, so that during every frame readout the AF driver is activated 2 rows below the bottom of the 4x4 array of AF windows. If af.windowPos and af.windowSize are such that the bottom of the

array is outside the frame, the value given to af.wakeUpLine by AF_SetSize is invalid (greater than frame height) and must be changed to something less than the frame height—otherwise AF does not work.

Auto Focus Mechanics Driver

The auto focus mechanics (AFM) driver comprises functions and variables that enable the AF driver to control different types of lens actuators through the general purpose I/O module (GPIO). The AFM driver provides the AF driver with a single, abstract, hardware-independent control interface hiding from it all details of GPIO programming and GPIO interactions with external devices. Each type of lens actuator requires different input signals and produces different response (lens movements and feedback signals), but can be controlled by the AF driver using only 4 variable function pointers (afm.vmt->pInit, afm.vmt->pSetPos, afm.vmt->pExecCmd, afm.vmt->pGetStatus) and three 1-byte variables (afm.curPos, afm.prePos, and afm.backlash) supplied by the AFM driver.

The function pointers are variable (by changing the value of afm.vmt) because they must point to different AFM driver functions depending on the current selection of lens actuator type indicated by the variable afm.type. By default, afm.type equals 0 and the pointers point to dummy functions that do nothing. The AF driver can safely use them, but using them has no effect on the GPIO - it generates no output signals. This is appropriate when no external AF hardware is connected to it. If there is a lens actuator to control, the AF driver can do so only after the afm.vmt is changed to give it access to the device-matching control functions in the AFM driver. Proper simultaneous change of afm.type and afm.vmt can be requested at any time from the MT9D111 sequencer, by first setting afm.type to desired new value plus 128 and then setting seq.cmd to 5.

The AFM driver supports three types of lens actuators, one of which (stepper motor type, afm.type=2) includes a variety of devices from different vendors, while the second (helimorph type, afm.type=1) encompasses just one actuator as of this writing. However, this actuator is representative of a broad category of devices that are controlled by writing simple commands through their serial interfaces and provide no feedback on their status. The AFM driver functionality developed to handle one such device can be partly reused to handle other similar devices. The third device (afm.type=3) supported is the AD5398 10-bit DAC IC.

Mode Driver

The mode driver reduces integration efforts by managing most aspects of switching between preview (mode A) and capture (mode B) modes. It remembers vital register values for each image acquisition mode and uploads these values to the appropriate registers upon each mode switch. In addition to remembering the register data, it also creates preloaded and custom gamma and contrast tables for each mode.

To change the mode driver parameters, upload the new values to the mode driver table (ID = 7). Upon the next mode change or sequencer REFRESH (CM_REFRESH) command, these mode driver values are uploaded to the appropriate sensor and system registers, and the image processing then reflects the new values (at the beginning of the next frame acquired).

To control the output image size, upload the dimensions to the mode driver variables: output width_A, output height_A, output width_B and output height_B. The mode driver automatically applies any appropriate downsizing filter to achieve this output image size as well as updating the FIFO output size when in JPG bypass (RAW) mode. It is important so set-up the imager to output an image equal to or larger than the target output image. It is also important for the crop window (crop_left_A/B and crop_right_A/B) to be equal to or larger than the target output image.

To upload a custom gamma table, upload the values to the appropriate mode driver locations (see Table 18, "Driver Variables-JPEG Driver (ID = 9)," on page 97), and select the gamma table type to be three (user defined) for the particular mode. If a contrast level is selected, it is applied to the user uploaded gamma table.

The driver contains settings for raw and output image size and pan for each mode. See variables such as mode.sensor_col_width_A/B, mode.sensor_row_height_A/B and mode.fifo_width_A/B, mode.fifo_height_A/B. Blanking and readout mode— the use of skip, binning, and 1ADC modes— is configured using sensor core registers R5:1-8:1 and R32:1, R33:1.

To change overall image brightness by changing ffset at output, set

- mode.y_rgb_offset_A for preview
- mode.y_rgb_offset_B for capture

To change output format, set

- mode.output_format_A for preview
- mode.output_format_B for capture

Similarly, the user can set special effects for each mode using mode.spec_effects_A/B.

Histogram Driver

The histogram driver continually works to reduce image flare and continually analyzes input image histogram and dynamically adjusts the black level, R59:1. When flare is present the histogram does not contain dark tones, causing the driver to subtract off a higher black level thus regaining the lost contrast. In certain situations the scene may contain no dark tones without flare. The histogram driver cannot distinguish this situation and alters the black level just the same, causing the image to have more contrast, which looks acceptable in many situations.

Variable hg.maxDLevel sets the maximum level that can be subtracted off the input data. Set this value to match lens flare percentage. For example, if a lens typically has a 5 percent flare, set this value to 0.05*1024 = 51. To disable flare subtraction in all modes, set this value to “0.” The maximum allowed value is 128. Read variable hg.DLevel to see the current subtracted value. hg.percent indicates how many percent of histogram dark tones need to be clipped. The recommended value is 0. The hg.DLevelBufferSpeed controls the speed of adjustment, 32 being fastest and 1 being the slowest.

Flicker Avoidance Driver

Indoor light sources powered from the wall AC power supply often exhibit fast oscillations. The oscillations are too fast to be seen; their frequency often being the frequency of the wall power supply. The oscillations interact with the sensor and can show in the image as standing or rolling horizontal bars. Such bars are visible when sensor integration time is not a natural multiple of the flicker period. Since there are two AC frequencies in common use, 50Hz and 60Hz, the imager integration time must match to ambient indoor flicker frequency to produce good images.

The MT9D111 can automatically detect the presence of flicker and adjust its integration time to avoid it.

To set up flicker avoidance program the following variables:

- fd.R9_step60 and fd.R9_step50 are flicker periods for 50 and 60Hz illumination expressed in row time, fd.R9_step60=Tflk/Trow. For 60Hz (1/120)/Trow and for 50Hz (1/100)/Trow. The light flicker frequency is twice the power frequency due to rectification.

- fd.search_f1/2_50 and fl.search_fl/2_60 are ranges of frequencies to look for:
  fd.search_f1_50=0.2*fd.R9_step60-1;
  fd.search_f2_50=0.2*fd.R9_step60+1;
  fd.search_f1_60=0.2*fd.R9_step50-1;
  fd.search_f2_60=0.2*fd.R9_step50+1
- fl.stat_min and fl.stat_max are detection thresholds; for flicker to be detected fl.stat_max continuous frames must contain the searched frequency at least fl.stat_min times

Other variables are as follows:

fd.windowPosH indicates current sampling window position. Variable fl.windowHeight indicate window size. Do not change these variables.

Flicker can operate in automatic and manual modes. Automatic mode is enabled by default, fd.mode[5] indicates current 50Hz/60Hz frequency selection. Make sure to set flicker detection bits in sequencer states. To choose manual mode, set fd.mode[7]=1. Bits fd.mode[6] is used for manual 50Hz/60Hz frequency selection. As frequencies are selected, the auto exposure integration time step is assigned the appropriate fd.R9_step50/60 variable value.

For the flicker avoidance driver to operate correctly, the sensor frame rate may not be exactly 15 fps or 30 fps. Frame rate must be tuned off the exact frame rate. This is a necessary condition to have the flicker bars rolling. For this reason the preview mode frame rate is typically deliberately set not to exactly match 15 fps or 30 fps. However, in capture mode, the frame rate is set exactly, assuming that the flicker has already been detected and compensated during preview.

JPEG Driver

The JPEG driver programs Huffman table and quantization table memories, sets up the MT9D111 to output JPEG compressed data, and handles error checking and handshaking with the host processor.

Usage

JPEG is enabled using mode.mode_config. For example, to enable compression for capture set mode.mode_congif=16. jpeg.qscale1/2/3, and specify the quantization factor to control compression ratio. Bigger number indicates more compression. mode.fifo* configure spoof and non-spoof output and specify FIFO output clock divider. If necessary, use jpeg.config for error handshaking with the host.

Table Programming

At power up initialization, the JPEG driver loads standard Huffman tables into Huffman memory. Scaled versions of standard luma and chroma quantization tables and are loaded into quantization memory. The calculation of the scaled quantization table is as follows:

Scaled_Q = (scaling_factor * standard_Q + 16) >> 5

Scaling factor is bit 6:0 of JPEG driver variables qscale1, qscale2, and qscale3. The standard quantization and Huffman tables are Tables K.1–K.4 of the ISO/IEC 10918-1 Specification. Host processor can override Huffman and quantization memory with any arbitrary Huffman and quantization tables through indirect register access.

If the host chooses to use the scaled standard quantization tables, bit 4 of JPEG driver variable config must be set to “1.” Scaling factor can be changed at any time. Whenever it is changed, bit 7 of the corresponding JPEG driver variable qscale must be set to “1,” and the new value takes effect in the next frame JPEG encoding.

Since the quantization memory stores 3 sets (luma and chroma) of quantization tables, the one that is used for the current frame JPEG encoding is indicated in bit 7:6 of jpeg.config (quantization table ID). Bit 5 of jpeg.config determines who is responsible for setting the quantization table ID. If it is "0," the host processor must program quantization table ID for every JPEG compressed frame (no need to reprogram it if subsequent JPEG frames use the same quantization tables). If it is "1," the JEPG driver sets quantization table ID to "0" at the start of JPEG capture (be it still or video) and automatically select next set of quantization tables for the subsequent frames when the current JPEG frame is unsuccessful.

Bit 7:6 of jpeg.config = 0, 1, and 2 indicates first, second, and third set of quantization tables, respectively.

Error Handling and Handshaking

When the capturing of a JPEG snapshot is unsuccessful, the JPEG driver can be configured to enable encoding subsequent frames until it is successful by setting bit 2 of jpeg.config to "1." For capturing JPEG video, MT9D111 always encodes subsequent frames no matter what value bit 2 of jpeg.config is set to.

If bit 1 of jpeg.config is "1," handshaking with the host processor at every erroneous JPEG frame is required. When JPEG status register indicates that there is an error in the current JPEG frame, JPEG encoding is stopped until the host processor sets bit 3 of jpeg.config to "1" to indicate it is ready to receive next JPEG frame. If the host processor does not respond to an erroneous JPEG frame within jpeg.timeoutFrames frames, JPEG capture is terminated. If bit 1 of jpeg.config is "0" or if there is no error in JPEG status register, no handshaking is required.

The handshaking mechanism is provided so that the host processor has sufficient time to handle the erroneous JPEG frame and react upon the error condition. For example, if the JPEG status register indicates FIFO overflow, the host processor should increase quantization value by changing quantization table scaling factor or selecting another set of the preloaded quantization tables. If spoof oversize error occurs, the host processor should increase the spoof frame size by programming registers R16 and R17 on IFP Register Page 2 and/or increase quantization value.

Start-Up and Usage

The start-up sequence consists of the following:

1. Power-up
2. Hardware reset
3. Configure and enable PLL
4. Configure pad slew rate
5. Configure preview mode
6. Configure and enable auto focus
7. Configure capture mode
8. Perform lock or capture

To start the part, power up power supplies, provide an input clock, and perform a hardware reset.

Power-Up

There are no specific requirements to the order in which different supplies are turned on. Once the last supply is stable within the valid ranges specified below, follow the hard reset sequence to complete the power-up sequence. In order to minimize leakage current, all the power supplies should be turned on at the same time.

Analog Voltage 2.8V for best image performance

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

I/O Voltage 1.7V–3.1V

Power-Down

Before powering down the sensor, it is recommended to bypass the PLL. Once this is completed, the input clock (CLKIN) can be turned off. After CLKIN is off, turn off all the power supplies as shown in Figure 33 on page 150. RESET# should also be LOW at this time.

Note: Turning the power supplies cannot be used as a method of achieving low power consumption. Power to the sensor needs to be provided as long as the system is active. For the lowest power consumption, please refer to the standby procedure in the MT9D111 Developer Guide or Technical Note TN0934.

Hard Reset Sequence

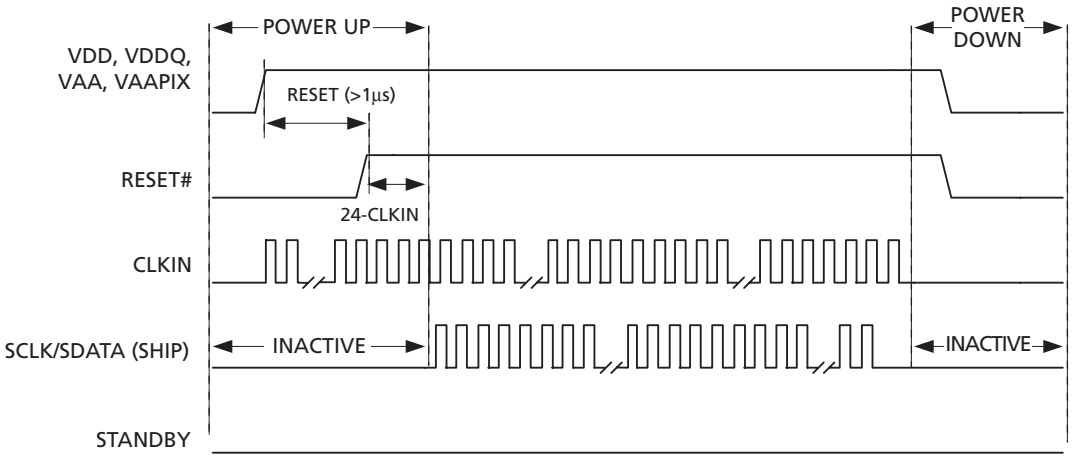
After power-up, a hard reset is required. Assuming all supplies are stable, the assertion of RESET# (active LOW) sets the device in reset mode. The clock is required to be active when RESET# is released. Hence, leaving the input clock running during the reset duration is recommended. After 24 clock cycles (CLKIN), the two-wire serial interface is ready to accept commands. Reset should not be activated while STANDBY is asserted.

A hard reset sequence to the camera can be activated by the following steps:

1. Wait for all power supplies to be stable and within specification.
2. Supply the sensor with an input clock.
3. Assert RESET# (active LOW) for at least 1µs.
4. De-assert RESET# (input clock must be running).
5. Wait 24 clock cycles before using the two-wire serial interface.

Please refer to Figure 33 on page 150 for the timing diagram.

Figure 33: Power On/Off Sequence



Notes:
1. For a safe RESET to occur, CLKIN should be running during RESET with STANDBY LOW, as shown in the sequence above.
2. After RESET# is HIGH, wait 24 CLKIN rising edges before the two-wire serial interface communication is initiated.
3. After the power-up sequence, the preview state is reached when the firmware variable seq.state (ID=1, Offset=4) is equal to 3. This transition time varies depending on the input clock frequency and scene conditions.
4. In order to go into the firmware standby state, go to capture mode (also known as context B), or execute the firmware REFRESH/REFRESH_MODE commands after the power-up sequence (the preview state [seq.state=3] must be reached first).

Soft Reset Sequence

A soft reset to the camera can be activated by the following procedure:

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

Note: No access to MT9D111 registers—both page 1 and page 2—is possible during soft reset.

Enable PLL

Since the input clock frequency is unknown, the part starts with PLL disabled. The default MNP values are for 10 MHz, with 80 MHz as target. For other frequencies, calculate and program appropriate MNP values. PLL programming and power-up sequence is as follows:

1. Program PLL frequency settings, R0x66-67:0
2. Power up PLL, R0x65:0[14] = 0
3. Wait for PLL settling time >150µs
4. Turn off PLL bypass, R0x65:2[15] = 0

Allow one complete frame to effect the correct integration time after enabling PLL.

Note: Until PLL is enabled the two-wire serial interface may be limited in speed. After PLL is enabled, the two-wire serial interface master can increase its communication speed.

Configure Pad Slew

Program the desired slew rate for DOUT, PIXCLK, FRAME_VALID, and LINE_VALID at the variables, mode.fifo_conf1/2_A/B. Program R10:1 with desired GPIO slew rate and slew rate for two-wire serial interface SDATA and SCLK.

Configure Preview Mode

The default preview image size is 800 x 600, running at up to 30 fps at 80 MHz internal clock. To change the default size, program mode driver variables mode.output_width_A and mode.output_height_A and issue a REFRESH command, seq.cmd=5.

For example, to configure 160x120 LCD RGB preview, program

- mode.output_width_A=160
- mode.output_width_B=120
- mode.out_format_A=0x20
- seq.cmd=5

Preview contrast, brightness, gamma, frame rate, and many other parameters can be loaded here as well. If known at this time, the user can also program capture parameters. If necessary, set up the AE/WB/AF lock command here.

Configure Capture Mode

When desired capture parameters are known (video, still, compression, and resolution), program the mode and other drivers correspondingly.

Perform Lock or Capture

See "Sequencer Driver" on page 140 for details.

Standby Sequence

Standby mode can be activated by two methods. The first method is to assert STANDBY, which places the chip into hard standby. Turning off the input clock (CLKIN) reduces the standby power consumption to the maximum specification of 100µA at 55°C. There is no serial interface access for hard standby.

The second method is activated through the serial interface by setting R13:0[2]=1 to the register, known as the soft standby. As long as the input clock remains on, the chip allows access through the serial interface in soft standby.

Standby should only be activated from the preview mode (context A), and not the capture mode (context B). In addition, the PLL state (off/bypassed/activated) is recorded at the time of firmware standby (seq.cmd=3) and restored once the camera is out of firmware standby. In both hard and soft standby scenarios, internal clocks are turned off and the analog circuitry is put into a low power state. Exit from standby must go through the same interface as entry to standby. If the input clock is turned off, the clock must be restarted before leaving standby. If the PLL is used to generate the master clock, ensure that the PLL is powered down during standby because it uses a relatively high amount of power. By default, R101:0[13] powers down the PLL when the chip enters standby mode. Turn on the PLL bypass (R101:0[15]=1) to prepare the PLL for standby.

To Enter Standby

1. Preparing for standby
    a. Issue the STANDBY command to the firmware by setting seq.cmd=3
    b. Poll seq.state until the current state is in standby (seq.state=9)
    c. Bypass the PLL if used by setting R101:0[15]=1

2. Preventing additional leakage current during standby
   a. Set R10:1[7]=1 to prevent elevated standby current. It controls the bidirectional pads DOUT, LINE_VALID, FRAME_VALID, PIXCLK, and GPIO.
   b. If the outputs are allowed to be left in an unknown state while in standby, the current can increase. Therefore, either have the receiver hold the camera outputs HIGH or LOW, or allow the camera to drives its outputs to a known state by setting R13:0[6]=1. R13:0[4] needs to remain at the default value of "0." In this case, some pads are HIGH while some are LOW. For dual camera systems, at least one camera has to be driving the bus at any time so that the outputs are not left floating.
   c. For each GPIO that is left floating (which are set as inputs by default), configure as outputs and drive LOW by the setting the respective bit to "0" in the GPIO variables 0x9078, 0x9079, 0x9070, and 0x9071 (accessed via R198:1 and R200:1). For example, if all GPIOs are floating inputs, the following settings can be used:
      i. R198:1=0x9078
      ii. R200:1=0x0000
      iii. R198:1=0x9079
      iv. R200:1=0x0000
      v. R198:1=0x9070
      vi. R200:1=0x0000
      vii. R198:1=0x9071
      viii. R200:1=0x0000
3. Check if other devices sharing the GPIO bus has conflicts with this arrangement
   a. If a GPIO configured as an input is not allowed to be set as output during standby, have the external source hold its output HIGH or LOW during standby.
4. Putting the camera in standby
   a. Assert STANDBY=1. Optionally, stop the CLKIN clock to minimize the standby current specified in the MT9D111 data sheet. For soft standby, program standby R13:0[2]=1 instead.

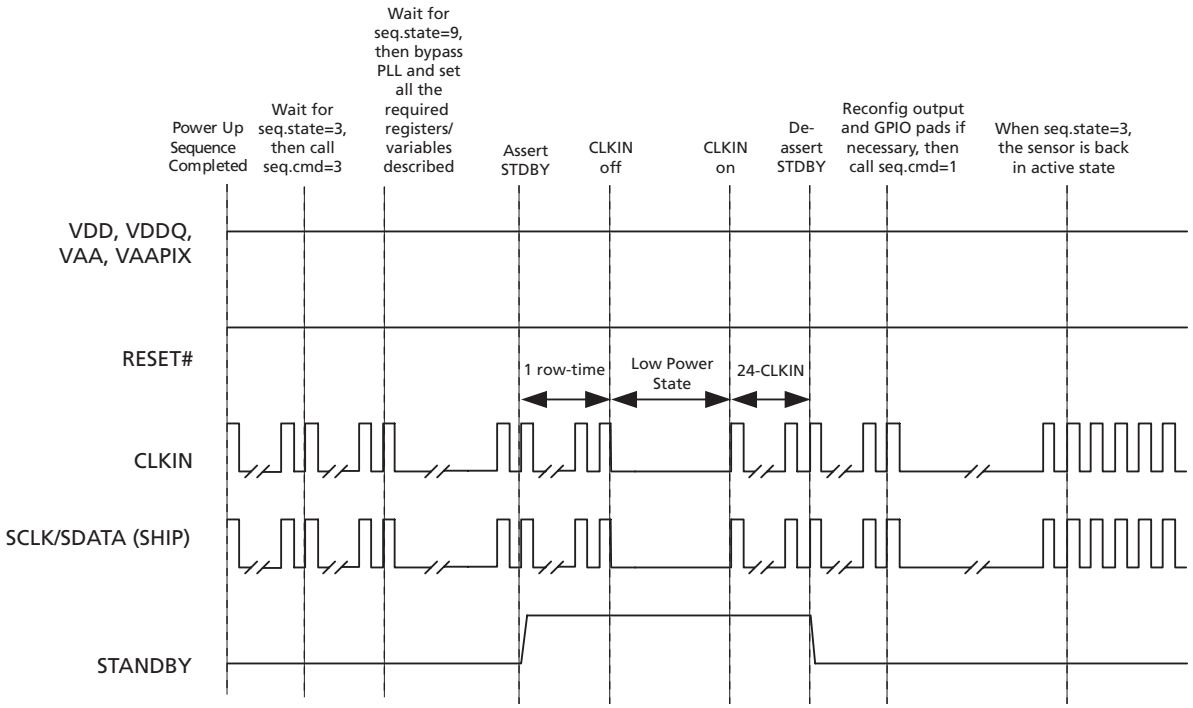
To Exit Standby

1. De-assert standby
   a. Provide CLKIN clock, if it was disabled when using STANDBY
   b. De-assert STANDBY=0 if hard standby was used. Or program R13:0[2]=0 if soft standby was used
2. Reconfiguring output pads
3. If necessary, reconfigure the GPIOs back to the desired state by GPIO variables 0x9078 and 0x9079. Also set R10:1[7]=0 if any GPIOs are used as inputs.
4. Issue a GO_PREVIEW command to the firmware by setting seq.cmd=1
5. Poll seq.state until the current state is preview (seq.state=3)

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

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

Figure 34: Hard Standby Sequence



Standby Hardware Configuration

While in standby, floating IO signals may cause the standby current to rise significantly. Therefore, it is recommended that the following signals be maintained high or low during standby and not floating: GPIO[11:0], DOUT[7:0], PIXCLK, FRAME_VALID, and LINE_VALID.

Output Enable Control

When the sensor is configured to operate in default mode, the DOUT, FRAME_VALID, LINE_VALID, and PIXCLK outputs can be placed in High-Z under hardware or software control, as shown in Table 38.

Table 38: Output Enable Control

Standby	R0x0D:0[4] (output_dis)	R0x0D:0[6]	Output State
0	0 (default)	0 (default)	Driven
1	0 (default)	0 (default)	High-Z
"Don't Care"	0 (default)	1	Driven
"Don't Care"	1	"Don't Care"	High-Z

The pin transition between driven and High-Z always occurs asynchronously. Output-enable control is provided as a mechanism to allow multiple sensors to share a single set of interface pins with a host controller.

There is no benefit in placing the pins in a High-Z while the part is in its low-power standby state. Therefore, in single-sensor applications that use STANDBY to enter and leave the standby state, programming R0x0D:0[6] = 1 is recommended.

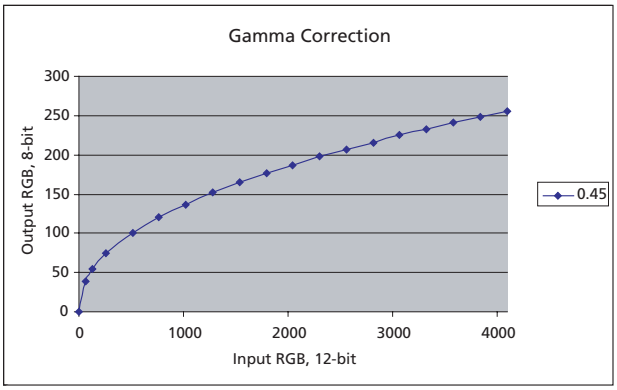
GPIO outputs can also be tri-stated. See "General Purpose I/O" on page 162 description for details.

Contrast and Gamma Settings

The MT9D111 IFP includes a block for gamma and contrast correction. A custom gamma/contrast correction table may be uploaded, or pre-set gamma and contrast settings may be selected.

The gamma and contrast correction block uses the following 12-bit input data points to form a piecewise linear transformation curve: 0, 64, 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2304, 2560, 2816, 3072, 3328, 3584, 3840, and 4095. These input points have been selected to provide more detail to the low end of the curve where gamma correction changes are typically the greatest. These points correspond to 8-bit output values that can be uploaded to the appropriate registers.

Figure 35: Gamma Correction Curve



For simplicity, predefined gamma and contrast tables may be selected, and the MT9D111 automatically combines these tables and upload them to the appropriate gamma correction registers.

The gamma and contrast tables may be selected at mode driver (ID = 7) offsets 67 and 68 (decimal) for mode A and mode B, respectively. The gamma settings are established at bits 0–2, and the contrast settings are established at bits 4–6.

The gamma setting values are shown in Table 39.

Table 39: Gamma Settings

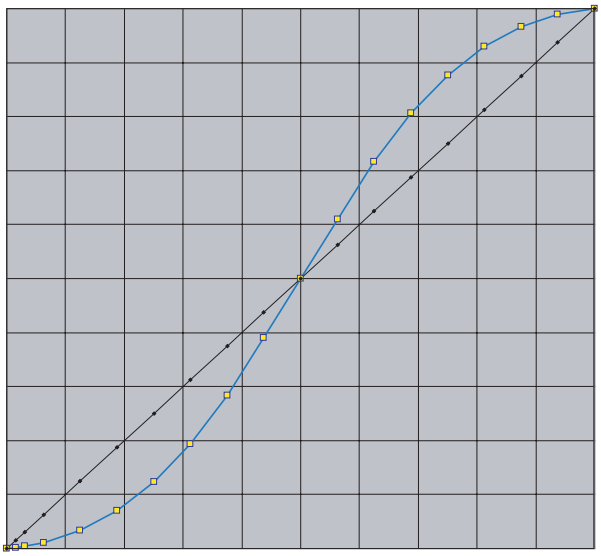
Gamma Setting	Definition
0	Gamma = 1.0 (no gamma correction)
1	Gamma = 0.56
2	Gamma = 0.45
3	Use user-defined gamma table

@=

The predefined contrast table values have been established by creating an "S" curve with highlight and shadow regions that blend smoothly with a linear midtone region. The slope and value of the highlight and shadow regions match the linear region at these

transitions. In addition, the slope of the "S" curve is zero at the top (white) and bottom (black) points. The slope of the linear region determines how much contrast is applied; more contrast corresponds to a higher, midtone linear slope.

Figure 36: Contrast "S" Curve



The contrast values are shown in Table 40.

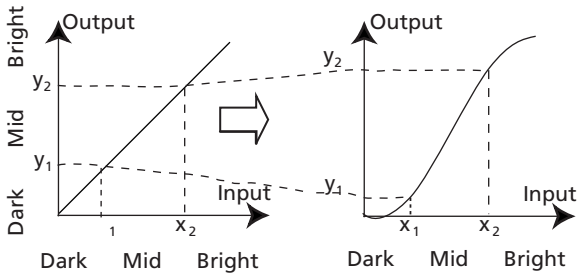
Table 40: Contrast Values

Contrast Setting	Definition
0	No contrast correction
1	Contrast slope = 1.25
2	Contrast slope = 1.50
3	Contrast slope = 1.75
4	Noise reduction contrast

The contrast curve function is applied to the gamma curve points used (whether the gamma curve points are predefined or user-uploaded).

S-curve is a function to correct image pixel values. When applied to pixel values, it typically compresses dark and bright tones, while stretching the midtones. Figure 37 shows how input tone range is remapped to output tone range. Suppose we categorize input pixel values as dark, midlevel, and bright. When no S-curve is applied, pixel values are unchanged. That is, dark tones $0..x_1$ map to same dark tones $0..x_1=y_1$, and midlevel maps to identical midlevel $x_1=x_2$, and bright maps to identical bright. When an S-curve is applied, the mapping is changed. In particular, midtones are stretched $(y_1-y_2)>(x_1-x_2)$, causing increase of contrast. Here $(y_1-y_2)/(x_1-x_2)$ is a measure of contrast. Value of 1 corresponds to no change; >1 and <1 indicate contrast increase and decrease, respectively. Dark tones are compressed, $y_1<x_1$, causing suppression of noise.

Figure 37: Tonal Mapping



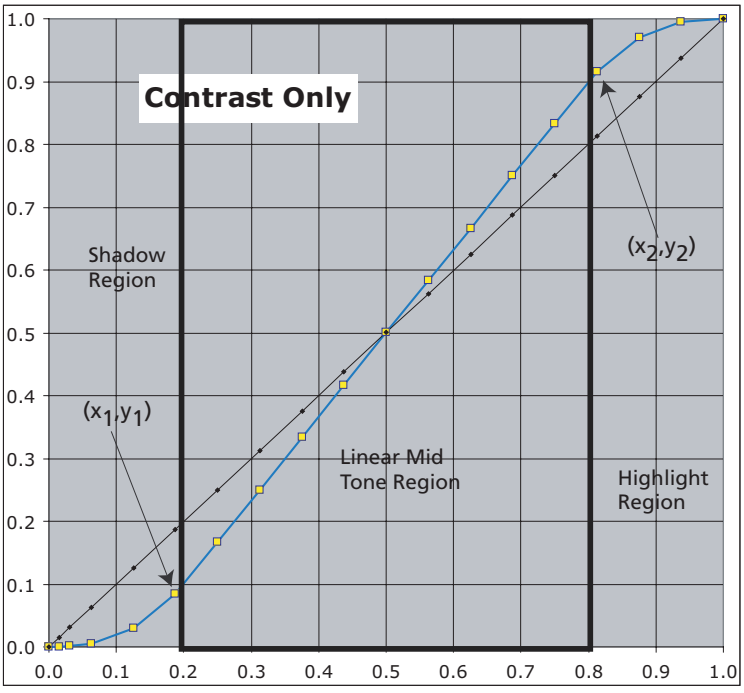
The contrast settings on the MT9D111 are implemented by applying an S-Curve function on to the data points of the gamma curve. These resulting points then replace the existing gamma curve points, and the image is processed using this new contrast-enhanced gamma curve. Identical S-curve is applied to all three RGB components.

The S-curve is created by joining a linear midsection (often with a slope greater than one) to curved sections for highlights and shadows. The following constraints apply to the overall curve to ensure a smooth curve appropriate for increased contrast:

1. The first/lowest point must be (0,0) and the last/highest point must be (1,1) (normalized).
2. The midsection and each of the end-curves must intersect on the same points.
3. The slope of the midsection and each of the end-curves must be equal at the intersection points.

The midsections a simple linear function of the form: y = mx + b. Each of the end-curves (shadow and highlights) must be a trinomial to satisfy all boundary conditions.

Figure 38: Contrast Diagram



Auto Exposure

Two types of auto exposure are available—preview and evaluative.

Preview

In preview AE, 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, 16 windows are combined in 2 segments: central and peripheral. Central segment includes four central windows. All remaining windows belong to peripheral segment. Scene brightness is calculated as average luma in each segment taken with certain weights. Variable ae.weights[3:0] specifies central zone weight, ae.weights[7:4] - peripheral zone weight.

The driver changes AE parameters (IT, Gains, and so on) to drive brightness (ae.CurrentY) to programmable target (ae.Target). Value of one step approach to target is defined by ae.JumpDivisor variable. Expected brightness is

Ynew = ae.CurrentY+(ae.Target-ae.CurrentY)/ ae.JumpDivisor.

To avoid unwanted reaction of AE on small fluctuations of scene brightness or momentary scene changes, the AE driver uses temporal filter for luma and gate around AE luma target. The driver changes AE parameters only if buffered luma outsteps AE target gates. Variable ae.lumaBufferSpeed defines buffering level.

32*Ybuf1=Ybuf0*(32-ae.lumaBufferSpeed)+Ycurr* ae.lumaBufferSpeed;

Values ae.lumaBufferSpeed=32 and ae.JumpDivisor=1 specify maximal AE speed.

Evaluative

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

Exposure Control

To achieve the required amount of exposure, the AE driver adjusts the sensor integration time R9:0, R12:0, gains, ADC reference, and IFP digital gains. To reject flicker, integration time is typically adjusted in increments of ae.R9_step. ae.R9_step specifies 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.

Exposure is adjusted differently depending on illumination situation.

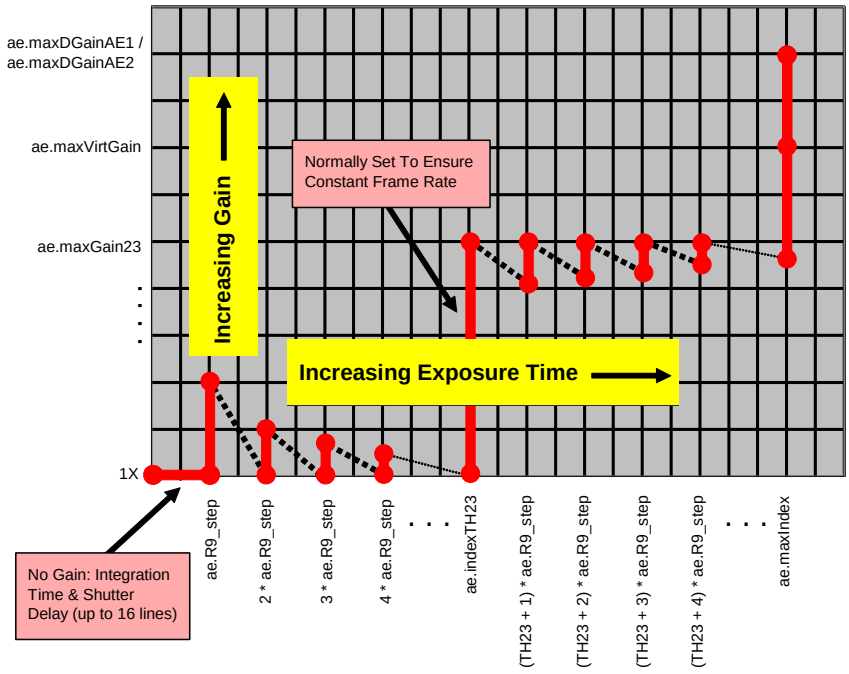
- In extremely bright conditions, the exposure is set using R12:0, R9:0 and analog gains. R12:0 is used to achieve very short integration times. In this situation, R9:0<ae.R9_step and flicker are not rejected.
- In bright conditions where R9:0>=ae., R9_step R9:0 is set as a natural factor of ae.R9_step. Analog gains are also used, but the green gain, also called virtual gain, does not exceed 2x. ae.minVirtGain limits minimal integration time and is expressed in flicker periods. ae.Index indicates the current integration time expressed in the same form.
- Under medium-intensity illumination, the integration time can increase further. For any given exposure, the best signal-to-noise ratio can be typically obtained by using the longest exposure and the smallest gain setting. However, a long exposure time can slow down the output frame rate if the former exceeds the default frame rate, R9:0 > R3:0 + R6:0 + 1. Integration ae.IndexTH23 specifies the breakpoint where AE scheme, giving preference to increasing the shutter width, is replaced with another scheme

giving preference to increase in gain. ae.maxGain23 specifies maximum allowed gain in this situation. ae.VirtGain indicates current green channel gain.

- In darker situations, the gain achieves ae.maxGain23 and the integration time is allowed to increase again up to ae.maxIndex.
- In yet darker situations, once the integration time achieves ae.maxIndex, the analog gain is allowed to increase up to ae.maxVirtGain.
- In very dark conditions, the digital IFP gains are allowed to increase up to ae.maxDGainAE1 and ae.maxDGainAE2.

ADC is used as an additional gain stage by adjusting reference levels. See ae.ADC* variables.

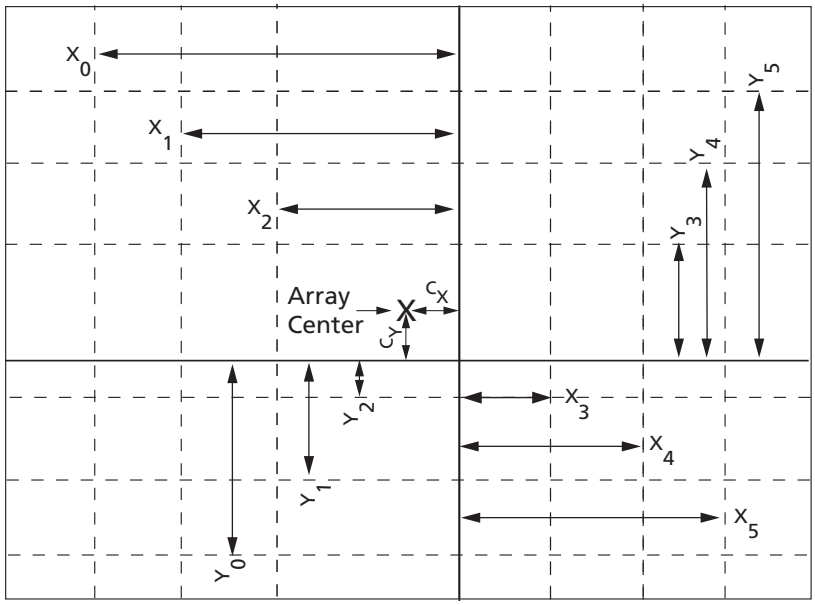
Figure 39: Gain vs. Exposure



Lens Correction Zones

In order to increase the precision of the correction function, the image plane is divided into 8 zones in each dimension. The coordinates of zone boundaries are referenced with respect to the lens center, C. Each boundary as well as C (Cx, Cy) coordinate is stored as a byte, which represents the coordinate value divided by 4. There always three boundaries to the left (top) of the center and three to the right (bottom) of the center. These boundaries apply uniformly for each color channel. However, the correction functions are programmable independently for each color component. Boundary and lens center positions are also valid for the preview mode. Figure 40 illustrates the lens correction zones.

Figure 40: Lens Correction Zones



Lens Correction Procedure

The goal of the lens shading correction is to achieve a constant sensitivity across the entire image area after the correction is applied. In order to accomplish this, each incoming pixel value is multiplied with a correction function F (x, y), which is dependent on the location of the pixel. The corrected pixel data, P_{OUT}, can be expressed in the following term:

$$P_{OUT}(x,y) = P_{IN}(x,y) * F(x,y) \quad \text{(EQ 1)}$$

Within each zone described above, the correction function can be expressed with a following equation, as follows:

$$F(x,y) = \phi(x,x^2) + \varphi(y,y^2) + k * \phi(x,x^2) * \varphi(x,x^2) * \varphi(y,y^2) - G \quad \text{(EQ 2)}$$

where

$$\phi(x,x^2) \text{ and } \varphi(y,y^2) \quad \text{(EQ 3)}$$

are piecewise quadratic polynomial functions that are independent in each x and y dimension

k and G are constants that can be used to increase lens correction at the image corners (k) and to offset the LC magnitude for all zones and colors (G)

The expressions

$$\phi(x,x^2) \text{ and } \varphi(y,y^2) \quad \text{(EQ 4)}$$

and are defined independently for each dimension. These can be expressed further as

$$\phi(x, x^2) = a_i x^2 + b_i x + c_i \quad \text{(EQ 5)}$$

$$\varphi(y, y^2) = d_i y^2 + e_i y + f_i \quad \text{(EQ 6)}$$

In order to implement the function F (x, y) for each zone the MT9D111 provides a set of registers that allow flexible definition of the function F (x, y). These registers contain the following parameters:

- operation mode R128:2
- zone boundaries and center offset R129:2-R135:2
- initial conditions of

$$\phi(x, x^2) \quad and \quad \varphi(y, y^2) \quad \text{(EQ 7)}$$

for each color (12-bit wide) R136:2-R141:2

- initial conditions of the first derivate of

$$\phi(x, x^2) \quad and \quad \varphi(y, y^2) \quad \text{(EQ 8)}$$

for each color (12-bit wide) R142:2-R147:2

- second derivatives of

$$\phi(x, x^2) \quad and \quad \varphi(y, y^2) \quad \text{(EQ 9)}$$

for each color and each zone (8-bit wide) R148:2-R171:2

- values for k(10-bit wide) and G(8-bit wide) in (2) for all colors and zones are specified in R173:2 and R174:2. Sign for k is specified in R R128:2.

There are x2 factor that can be applied to the second derivatives inside each zone (R172:2) if correction curvature in the particular zone is to be increased. There are also global x2 factors for all zones in X and Y direction located in R128:2. The first derivative (for both X and Y directions) can be divided by a number (devisor) specified in R128:2 before it is applied to the F function. Higher numbers result in more precise curve (full-scale expanse).

Color Correction

Color correction in the color pipeline is achieved by

1. Apply digital gain to raw RGB, R106:110:1
2. Subtract second black level D from raw RGB, see R59:1
3. Multiply result by CCM, R96-102:1

4. Clip the result

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = CLIP\left(\begin{vmatrix} C_{11} C_{12} C_{13} \\ C_{21} C_{22} C_{23} \\ C_{31} C_{32} C_{33} \end{vmatrix} \bullet \begin{vmatrix} R_{raw} * G_R - D \\ G_{raw} * G_G - D \\ B_{raw} * G_B - D \end{vmatrix} \right)$$

MCU controls both CCM and D; see "Auto White Balance Driver" on page 143 and "Histogram Driver" on page 146.

Decimator

In order to fit image size to customer needs, the image size of the SOC can be scaled down. The decimator can reduce image to arbitrary size using filtering. The scale-down procedure is performed by transferring an incoming pixel from the image space into a decimated (scaled down) space. The procedure can be performed in both X and Y dimension. All the standard formats with resolution lower than 2 megapixels as well as customer specified resolutions are supported. Transfer of pixel from the image into the decimated space is done in the following way, which is the same in X and Y directions:

Each incoming pixel is split in two parts. The first part (P1) goes into the currently formed output-space pixel while part two (P2) goes to the next pixel. P1 could be equal or less than value of the incoming pixel while P2 is always less. These two parts are obtained by multiplying the value of incoming pixel by scaling factors f1 and f2, which sum is always constant for the given decimation degree and proportional to X1/X0 where X1 is size of the output image and X0 is the size of the input image. It is denoted as "decimation weight." Coefficients f1 and f2 are calculated in the microcontroller transparently for user based on the specified output image size and mode of SOC operation. At large decimation degrees, several incoming pixels may be averaged into one decimated pixel. Averaging of the pixels during decimation provides a low pass filter, which removes high-frequency components from the incoming image, and thus avoids aliasing in the decimated space. The decimator has two operational modes—normal and high-precision. Since the intermediate result for Y decimated pixels has to be stored in a memory buffer with certain word width, there is a need for additional precision at larger decimation degrees when scaling factors are small. This is done by increasing the number of digits for each stored value when decimation is greater than 2.

General Purpose I/O

Introduction

Actuators used to move lenses in AF cameras can be classified into several categories that differ significantly in their requirements for driving signals. These requirements vary also from one device to another within each category. The MT9D111 has been designed to meet the needs of many different lens actuators without having a lot of hardware resources dedicated solely to that purpose. Internal resources that MT9D111 brings together to control a lens actuator can be, and at least in part are, used for many other tasks, which amply justifies their presence on the chip even when no AF support is required from it. This is particularly true of the embedded 6811 microcontroller (MCU) with associated memory, but also of the general purpose input/output module (GPIO), whose description is given below.

The GPIO is, in essence, a programmable digital waveform generator with 12 individually controllable output pads (GPIO0 through GPIO11), a separate power supply pad (VDDGPIO), and a separate clock domain. By default, the GPIO clock domain is connected to the master clock, but it can be disconnected to save power, by writing 0 to R11:1[7]. The maximum rate at which the GPIO outputs can be toggled is 1/2 of the master clock frequency. In other words, when the master clock frequency is 80 MHz, the GPIO can change the state of its output pads every 25ns. This maximum rate is attainable only when the GPIO outputs pre-programmed periodic waveforms—a discussion of these and of differently generated arbitrary output patterns follows in the next two sections.

Since some lens actuators provide feedback signals that can be used to ascertain their position, direction of motion, etc., the MT9D111 has the capability to sense such signals via digital and analog inputs. All of the GPIO output pads can be individually reversed to become high-impedance digital inputs. Section “Digital and Analog Inputs” on page 144 explains how to change the direction of the GPIO pads and how to sense both digital and analog input signals.

The GPIO is programmed via 77 8-bit registers mapped to hexadecimal addresses 0x1070–0x10B6 and 0x10B8–0x10BD in the memory space of the MCU. Sixty-five of these registers and two additional registers with hexadecimal addresses 0x10BE and 0x10BF return information about GPIO status when read. Like driver variables, the GPIO registers can be accessed by an external host processor through registers R198:1 and R200:1. Since the two-wire serial interface is relatively slow, the host processor can never access the GPIO registers as fast as the embedded MCU. In GPIO applications requiring exact output timing and/or fast response to input signals, all time-critical writes and reads to the GPIO registers should be done by the MCU. If the only task of the GPIO is to move an AF lens, Micron strongly recommends leaving the control of the GPIO entirely to AFM driver (ID = 6), a part of MT9D111 firmware dedicated to that task. MT9D111 users have the option to substitute their own code for the AFM driver or part of it, if it does not meet their needs.

GPIO Output Control Overview

There are two ways to control voltages on the GPIO output pads. One way, direct but not guaranteeing high timing precision, is to set or clear bits in GPIO_DATA_L and GPIO_DATA_H registers. Both the MCU and the host processor can do it by writing to the hexadecimal addresses 0x1070–0x1077. The first two of these provide normal, "what-you-write-is-what-you-get" access to the registers, while the remaining six provide selective access to individual register bits. This selective access facilitates changing voltages on some GPIO output pads without affecting the state of other pads—an output manipulation expected to be routine. For example, to change voltage on the output pad

GPIO3, the user only has to write 8 to the address 0x1073 (also known as register GPIO_OUTPUT_TOGGLE_L). To do the same by writing to the register GPIO_DATA_L, the user must know in advance the state of its third bit and all other bits corresponding to output pads.

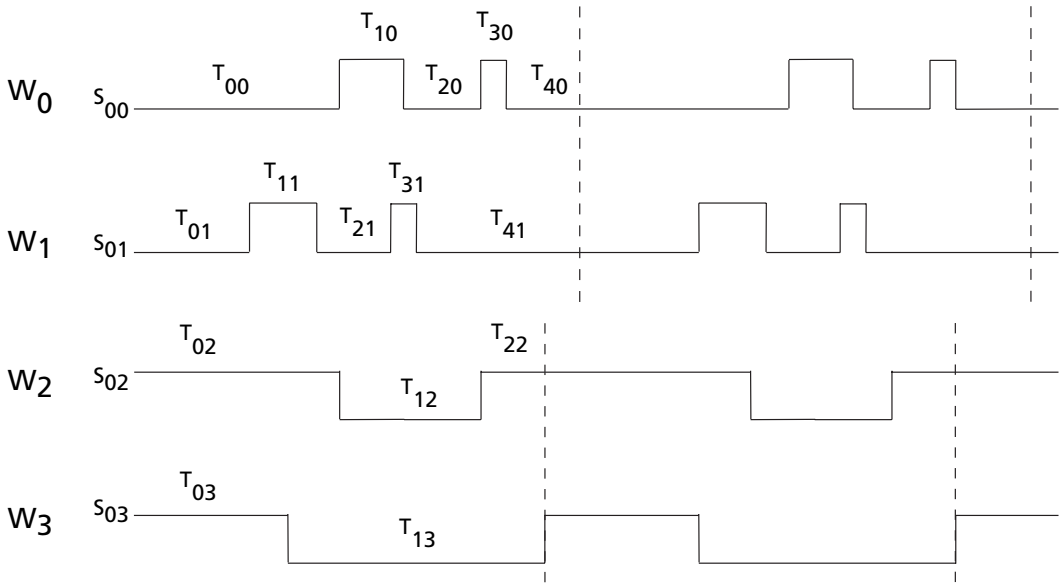
In general, writing a positive number to one of the GPIO_OUTPUT_* registers has the following effect: the GPIO output pads corresponding to ones in the binary representation of the number are toggled, set, or cleared, while the output pads corresponding to zeros and all input pads are left alone (masked). Once the GPIO outputs and the GPIO_DATA_* registers are updated, all bits in the GPIO_OUTPUT_* registers are automatically cleared.

The second way to obtain a desired output from the GPIO is to program into it a set of periodic waveforms, initialize their generation, and optionally monitor its progress. The advantage of using this way is that the GPIO, once programmed and activated, generates the desired waveforms on its own, without waiting for any external stimuli, and therefore with the best attainable timing accuracy. It is possible to override, suspend or abort this autonomous waveform generation by writing to appropriate GPIO registers, but no register writing is necessary for the GPIO to continue. If necessary, the GPIO can notify the MCU about reaching certain points on the waveform generation timeline, e.g., the first rising edge or end of a selected waveform. See "Notification Signals" on page 165 for more details.

Waveform Programming

A large subset of the GPIO registers is dedicated to programming periodic waveforms. In designing this subset, the main concern has been to meet the requirements of the most demanding AF actuators known to date. All registers belonging to this subset have names starting with GPIO_WG_, where WG stands for waveform generator. The GPIO_WG_* registers allow one to individually specify up to 8 waveforms, to be output through the GPIO[7:0] pads. It is not possible to output preprogrammed waveforms through the GPIO[11:8] outputs. These outputs are controlled only by the GPIO_DATA_H and GPIO_OUTPUT_*_H registers.

Figure 41: Examples of GPIO-Generated Waveforms



Of the four waveforms depicted in Figure 41, the first two are examples of the most complex waveforms that the GPIO can generate. Periods of these waveforms consist of five different time intervals (subperiods), the first four of which end with a transition to an opposite state (LOW-to-HIGH or HIGH-to-LOW). Each waveform, W_p (p = 0, 1), is completely described by its initial state, S_{0p}, the lengths of its five subperiods, T_{ip} (i = 0,…, 4), and the number of periods (repetitions) from the start to the end, N_p. The simpler waveforms W2 and W3 can be described using the same set of numbers, only with some of the subperiod lengths equal to 0. A valid waveform description must include $N_p > 0$ and at least one $T_{ip} > 0$, (i.e. it must set a nonzero duration for the waveform). Just one $T_{ip} > 0$ gives a constant function of time, $W_p(t) = S0p$. Generating such a waveform means keeping a particular GPIO output at LOW or HIGH for the specified time. To toggle an output between LOW and HIGH, one has to assign to it at least two nonzero T_{ip} values.

The S_{0p} values are set in the GPIO_DATA_L register. There are two ways to store the N_p and T_{ip} values in the GPIO registers GPIO_WG_N* and GPIO_WG_T*. The first is to allocate 8 bits for each value, which allows one to store values for eight waveforms. The second is to allocate 16 bits per value. When this is done across the board, one can define only up to four waveforms, to be generated at the GPIO0, GPIO2, GPIO4, and GPIO6 pads. However, the GPIO allows users to select "8-bit counter mode" or "16-bit counter mode" individually for each of the following pairs of outputs: GPIO[1:0], GPIO[3:2], GPIO[5:4], and GPIO[7:6].

These pairs are put in the 8- or 16-bit counter mode by setting bits [4:7] in register GPIO_WG_CONFIG to 0 or 1, respectively. The term counter mode is used here because these bits control the width of counters used in waveform generation. What is described above as allocating 8 or 16 bits per N_p or T_{ip} value in the GPIO_WG_N* and GPIO_WG_T* registers, is in fact done by changing counter widths and the way register values are loaded into the counters. Also, strictly speaking, there is a proportionality, not equality, relation between T_{ip} values and single or coupled GPIO_WG_T* register settings that represent them. Hence, the statement that T_{ip} values are stored in 8-bit or16-bit "cells" is a bit inaccurate. Perhaps we should say "encoded" instead of "stored."

In any case, the N_p values occupy up to 8 registers (GPIO_WG_N0 through GPIO_WG_N7). Writing an invalid $N_p = 0$ to any of these registers is interpreted as setting this particular N_p to infinity. The T_{ip} values are fully encoded in 42 registers named GPIO_WG_T*, GPIO_WG_CLKDIV, and GPIO_WG_CLKDIV_SEL. This last two registers contain, respectively, two 4-bit settings for two clock dividers and eight 1-bit switches assigning one or the other divider to each of the GPIO[7:0] pads. Each clock divider divides the GPIO clock frequency by 2^{d+1}, where d is its 4-bit setting. The GPIO_WG_T* registers contain natural numbers obtained by dividing the T_{ip} values by master clock period and by the appropriate power of 2. For example, suppose that the master clock period is 12.5 ns. If one decides to use the registers (and counters) in the 16-bit mode and sets the clock divider for the GPIO2 output to $2^{2+1} = 8$, one has to write 39 to GPIO_WG_T03 and 16 to GPIO_WG_T02 to get $T_{02} = 1$ ms. To obtain approximately the same T_{02} in the 8-bit mode with GPIO2 clock divider set to $2^{8+1} = 512$, one has to set the GPIO_WG_T02 to 156 (since 1ms/12.5ns/512 = 156.25).

Bit values in registers GPIO_WG_FRAME_SYNC, GPIO_WG_STROBE_SYNC and GPIO_WG_CHAIN determine the conditions that must be met for waveform generation to begin on each of the GPIO[7:0] pads. It always has to be enabled first by clearing an appropriate bit in GPIO_WG_SUSPEND register. Depending on selections made in the GPIO_WG_*_SYNC and GPIO_WG_CHAIN registers, the enabling is the signal to either start waveform generation immediately or on one of the following events: next falling edge of FRAME_VALID, next raising edge of STROBE or end of waveform generation on another pad. No more than one of these events should be chosen to trigger waveform generation on each particular pad. If more than one event is selected, only selection made in the highest priority register has an effect. The order of priority is, from highest to lowest, GPIO_WG_CHAIN, GPIO_WG_FRAME_SYNC, GPIO_WG_STROBE_SYNC.

Waveform generation at any pad can be suspended and resumed at will using the GPIO_WG_SUSPEND register. Suspending is like stopping the time on a particular pad. If generation on other pads continues in the meantime, synchronization between the suspended waveform and others are lost. Register GPIO_WG_RESET contain reset bits for the GPIO[7:0] outputs. Setting any of them to 1 aborts ongoing waveform generation at the corresponding pad and resets the counters used in it. The bit must be cleared before the waveform generation can resume.

Notification Signals

The GPIO can send signals to the MCU to notify it about two types of events: the end of waveform generation at a particular output pad (GPIO[7:0]) or a transition of interest (LOW-to-HIGH or HIGH-to-LOW) on a particular input or output pad (GPIO[11:0]).

The GPIO uses two kinds of notification signals in parallel: on each event of interest, it sends a wake-up signal to the MCU and it sets to 1 the appropriate bit in register GPIO_NS_STATUS_L or GPIO_NS_STATUS_H. The bit remains set until acknowledged by writing 1 to it. The wake-up signal has no effect unless the MCU is in sleep mode.

To enable notification signals from GPIO, some bits in register GPIO_NS_MASK_L and/or register GPIO_NS_MASK_H must be set to 0. By writing to the corresponding bits in registers GPIO_NS_TYPE, GPIO_NS_EDGE_L, and GPIO_N_EDGE_H, one can choose events that triggers the signals.

Digital and Analog Inputs

All GPIO pads are configurable as high-impedance digital inputs. Setting or clearing bits in the GPIO_DIR_* registers turns the corresponding pads into outputs or inputs, respectively. The logical state of each input pad is mirrored by the state of the corresponding bit in the GPIO_DATA_* registers, enabling the MCU or external host processor to receive digital feedback.

One of the 10-bit ADCs in the MT9D111 sensor core is available to sample external voltage signals (0.1 to 1.0V) during horizontal blanking periods, when it does not digitize sensor signal. The external signals that need to be sampled must be connected to AIN1, AIN2, and/or AIN3 input pads. By default, these are test pads that give access to different points in the sensor analog signal chain. To connect them directly to the ADC and enable signal sampling during horizontal blanking, R0xE3[15] must be set to 1. When this bit is set, 10-bit values of digitized AIN3, AIN2, and AIN1 signals are put in registers R0xE0:0, R0xE1:0, and R0xE3:0, respectively. The maximum signal sampling rate provided by this scheme is about 36000 samples per second.

GPIO Software Drivers

It is likely that some MT9D111 users wants to develop their own software for controlling the GPIO and, through it, their own devices. All MT9D111 firmware, in particular the AFM driver controlling the GPIO, has been designed to facilitate user modifications and additions. Public driver methods (functions) have been made easily replaceable by means of jump tables, also referred to as virtual method tables (VMT). Public firmware variables include pointers to these tables, which in turn include pointers to driver methods that users may want to replace with their own. To replace one or more public method of, say, the AFM driver, a user must do the following:

1. Load code containing replacement methods into the MCU RAM memory
2. Create in the RAM a copy AFM driver VMT to replace the original VMT
3. Populate the replacement VMT with pointers to the replacement methods and to those original methods that need not be replaced
4. Change the pointer to the AFM driver VMT included among the public variables of the driver so that it points to the replacement VMT.

The public VMTs also make it easy for the users to call firmware methods from their own code. Utilizing these methods may reduce user code size and shorten development time.

A detailed discussion of driver requirements of various lens actuators will be added to this document later.

Auto Focus

Algorithm Description

The AF algorithm implemented in the MT9D111 seeks to maximize sharpness of vertical lines in the sensor's image output by guiding an external lens actuator to the position of best lens focus. The algorithm's implementation has a hardware component called 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, 1-dimensional position variable, leaving the translation of its changes into physical lens movements to a separate AF mechanics (AFM) driver. The AF algorithm relies on the AFM driver and the GPIO to generate digital output signals needed to move different lens actuators. The AFM driver must also 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.

Line sharpness measurements are performed continuously (in every frame) by the FME, which is a programmable edge-filtering module in Image Flow Processor (IFP). The FME convolves two pre-programmed 1-dimensional digital filters with luminance (Y) data it receives row by row from the color interpolation module. In every interpolated image, the pixels whose Y values are used in the convolution form a rectangular block that can be arbitrarily positioned and sized, and in addition divided into up to 16 equal-size sub-blocks, referred to as AF windows or zones. The absolute values of convolution results are summed separately for each filter over each of the AF windows, yielding up to 32

sums per frame. As soon as these sums or raw sharpness scores are computed, they are put in dedicated IFP registers, as are Y averages from all the AF windows. The AF driver reduces these data to 1 normalized sharpness score per AF window, by calculating for each window the ratio (S1+S2)/<Y>, where <Y> is the average Y and S1 and S2 are the raw sharpness scores from the 2 filters multiplied by 128. Programming of the filters into the MT9D111 includes specifying their relative weights, so each ratio can be called a weighted average of two equally normalized sharpness scores from the same AF window. In addition to unequal weighting of the filters, the AF driver permits unequal weighting of the windows, but window weights are not included in the normalized sharpness scores, for a reason that will soon become clear.

There are several motion sequences through which the AF driver can bring a lens to best focus position. An example sequence is depicted in Figure 42. All these sequences begin with a jump to a preselected start position, e.g. the infinity focus position. This jump is referred to as the first flyback. It is followed by a unidirectional series of steps that puts the lens at up to 19 preselected positions different from the start position. This series of steps is called the first scan.

Before and during this scan, the lens remains at each preselected position long enough for the AF driver to obtain valid sharpness scores. Typically, the time needed is no longer than 1 frame, but there is an option to skip 1 frame before the AF driver grabs the scores, so the total time spent at each position can reach 2 frames. The timing of lens movements between the preselected positions is lens-actuator-dependent and not controlled by the AF driver. Though the AF driver gives commands to move the lens, it is the AFM driver that takes care of their execution and determines how soon after each command the AF driver gets a signal to proceed. All inclined sections of the lens position plot in Figure 42 are therefore of unknown duration—unless the AF algorithm discussion is narrowed to a specific use case.

Figure 42: Search for Best Focus

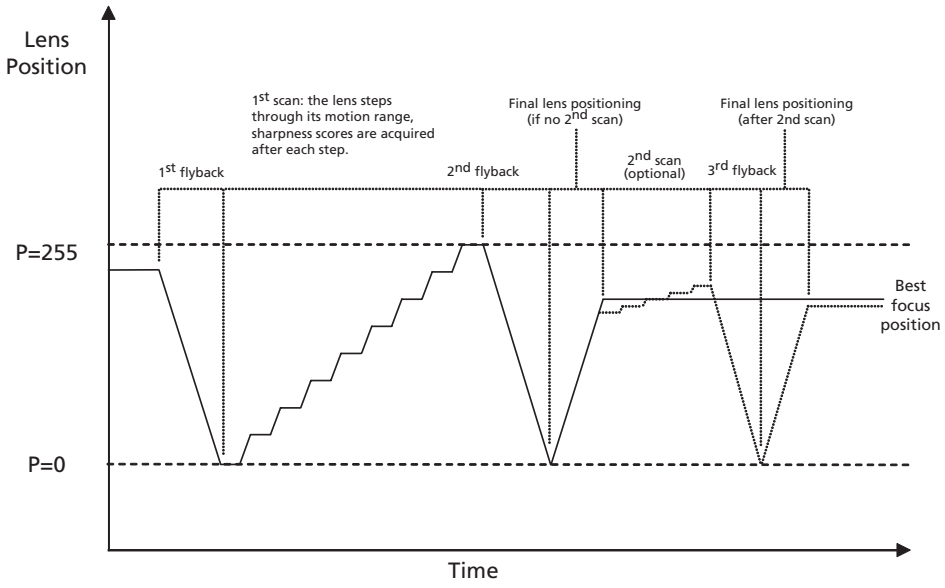


Figure 42 shows lens movements during dual/triple-flyback auto focusing sequence. The depicted sequence is just an example and can be changed in a number of ways. Second scan, as well as second and third flyback are optional—final lens positioning can be

a direct jump from last position tried in a scan to best focus position. Number of steps in each scan, lens positions stepped through during the first scan, and step size in the second scan are all individually programmable.

The first normalized score from each AF window, acquired at the start position, is stored as both the worst (minimum) and best (maximum) score for that window. These two extreme scores are then updated as the lens moves to subsequent positions and a new maximum position is memorized at every update of the maximum score. In effect, the preselected set of lens positions is scanned for maxima of the normalized sharpness scores, while at the same time information needed to validate each maximum is being collected. This information is in the difference between the maximum and the minimum of the same score. A small difference in their values indicates that the score is not sensitive to the lens position and therefore its observed extrema are likely determined by random noise. On the other hand, if the score varies a lot with the lens position, its maximum is much more likely to be valid, i.e. close to the true sharpness maximum for the corresponding AF window. Due to these considerations, the AF driver ignores the maxima of all sharpness scores whose peak-to-trough variation is below a preset percentage threshold. The remaining maxima, if any, are sorted by position and used to build a weight histogram of the scanned positions. The histogram is build by assigning to each position the sum of weights of all AF windows whose normalized sharpness scores peaked at that position. The position with the highest weight in the histogram is then selected as the best lens position.

This method of selecting the best position may be compared to voting. The voting entities are the AF windows, i.e. different image zones. Depending on the imaged scene, they may all look sharp at the same lens position or at different ones. If all the zones have equal weight, the lens position at which a simple majority of them looks sharp is voted the best. If the weights of the zones are unequal, it means that making some zones look sharp is more important than maximizing the entire sharp-looking area in the image. If there are no valid votes, because sharpness scores from all the AF windows vary too little with the lens position, the AF driver arbitrarily chooses the start position as the best.

Figure 43, Figure 44 and Figure 45 illustrate selection of best lens position when there are several objects in imaged scene to focus on, each at a different distance from the lens. Each lens position bringing one or more of these objects into sharp focus within the AF window grid can be potentially voted the best. The actual result of the vote is determined by the extent and texture of each object and the weights of the overlying AF windows.

What happens after the first scan and ensuing selection of best position is user-programmable—the AF algorithm gives the user a number of ways to proceed with final lens positioning. The user should select a way that best fits the magnitude of lens actuator hysteresis and desired lens proximity to the truly optimal position. Actuators with large, unknown or variable hysteresis should do a second flyback, i.e. jump back to the start position of the first scan, and then either retrace the steps made during the scan or directly jump to the best of the scanned positions. Actuators with constant hysteresis (like gear backlash) can be moved to that position directly from the end position of the scan—the AF algorithm offers an option to automatically increase the length of this move by a pre-programmed backlash-compensating step. Finally, if the first scan is coarse relative to the positioning precision of the lens actuator and depth of field of the lens, an optional second fine scan can be performed around the lens position selected as best after the first scan.

The second scan is done in the same way as the first, except that the positions it covers are not preset. Instead, the AF algorithm user must preset step size and number of steps for the second scan and enable its execution by setting the appropriate control bit in one of AF driver variables. Finding this bit equal to 1 at the end of the first scan, the AF driver

calculates lens positions to be tried in the second scan from its 2 user-set parameters and the position found best in the first scan. The calculation takes into account where that position is relative to the limits of the lens motion range and how it would be reached if the second scan were not enabled. If the user-selected way to reach it includes the second flyback, the AF driver assumes that the start position of the second scan must likewise be reached not directly from end position of the first scan, but via logical position 0, the default start position of that scan. An extra zero is therefore put at the beginning of the list of positions calculated for the second scan—unless this list already starts with logical position 0. If the second flyback is not enabled, no extra zero is prepended to the list. In every case, the list is then appended to the list of positions already scanned in the first scan. The combined list cannot have more than 20 entries, due to fixed 20-byte size of memory buffer used by the AF driver to store lens positions. This means that the first scan of, say, 15 positions can be followed by a flyback to 0 and second scan of no more than 4 non-zero positions or, alternatively, a second scan of up to 5 non-zero positions if the second flyback is not enabled. In both cases, the 2 unidirectional scans can be also seen as a single scan with 2 changes of direction. If the lens actuator has significant hysteresis, the effect of those changes should be carefully considered. The only way to alleviate it is to do the flyback to 0 prior to the second change.

The second scan is always followed by the user-selected final positioning sequence that in the absence of the second scan would follow the first scan, e.g. a flyback to the start position of the latter and a jump to the position found best in the former.

Figure 43: Scene with Two Potential Focus Targets at Different Distances from Camera

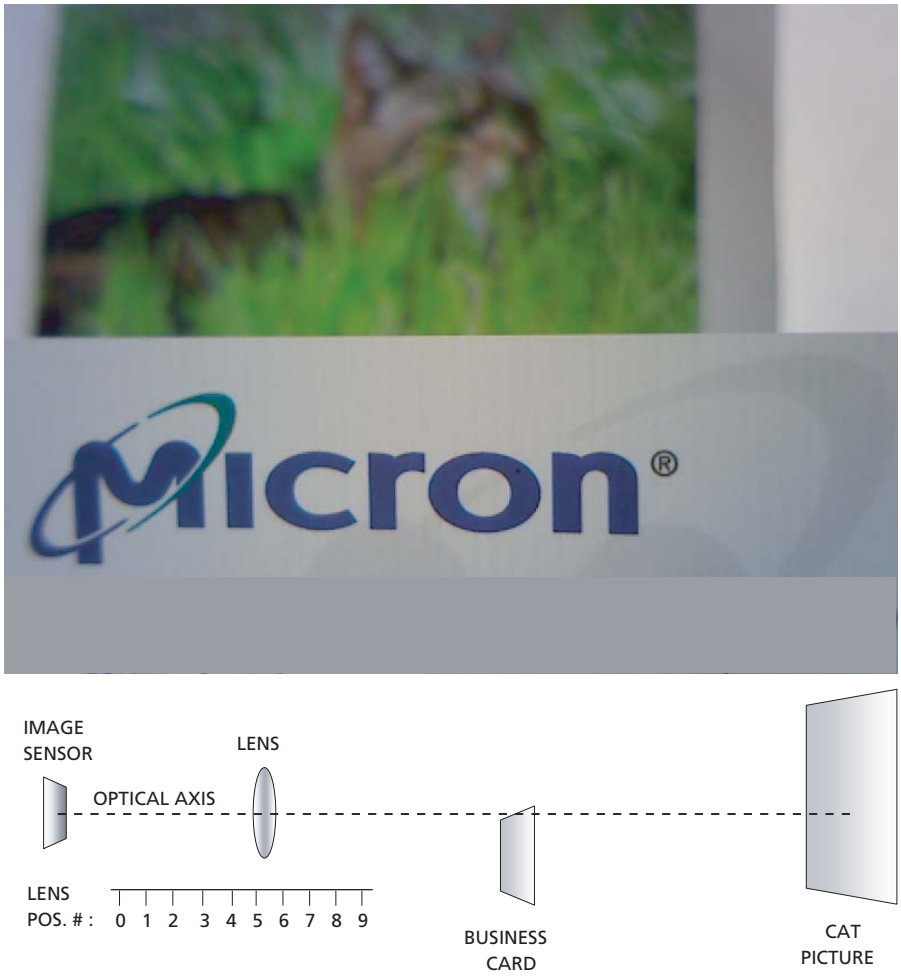


Figure 43 shows a simple scene with two potential focus targets: a business card in front and a picture of a cat far in the background. Distances in the diagram are not to scale. Red and yellow rectangles in the middle of the image represent 16 AF windows, each of which yields a separate Y-normalized sharpness score. Sharpness scores from the lower 8 windows are highest when the business card is in focus, which happens when the lens is at position 5. Scores from the upper 8 windows peak when the lens is at position 0. See Figure 45.

Figure 44: Dependence of Luminance-Normalized Local Sharpness Scores on Lens Position

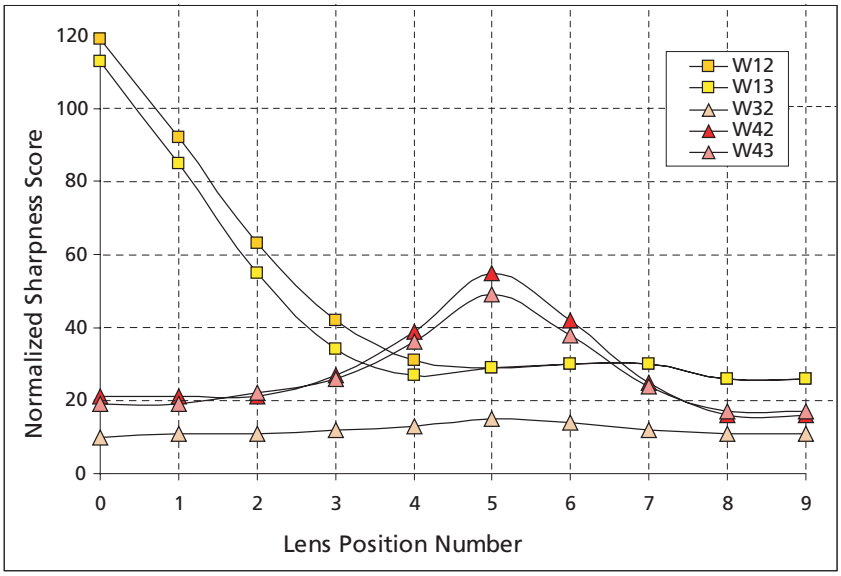


Figure 44 shows luminance-normalized sharpness scores from AF windows W12, W13, W32, W42, and W43 in Figure 43. Lens position 0 is the position of best focus for windows W12 and W13, while windows W42 and W43 are in sharpest focus at lens position 5. Relatively featureless window W32 is in sharpest focus at the same position, but it is difficult to determine this from its sharpness score, which varies very little with lens position.

Figure 45: Example of Position Weight Histogram Created by AF Driver

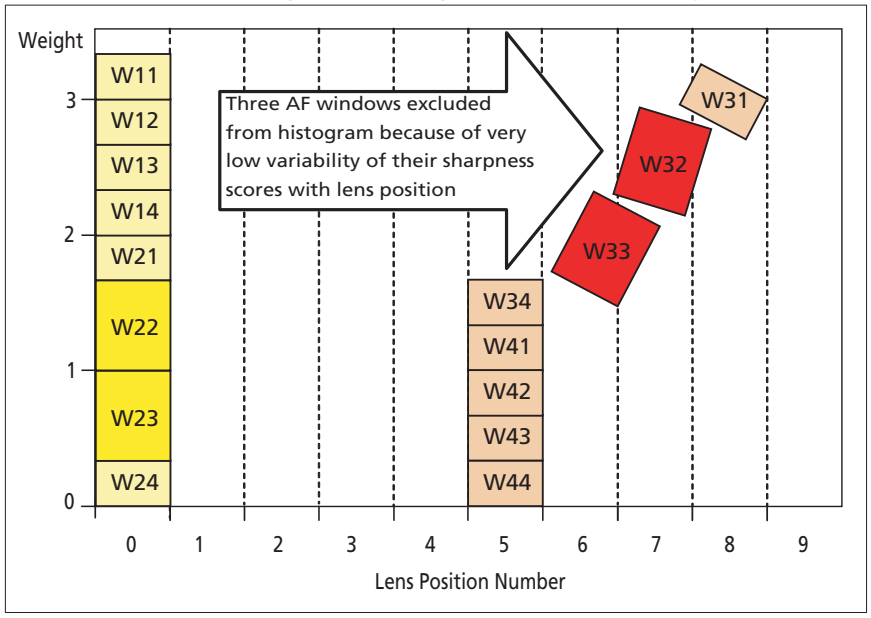


Figure 45 shows an example of position weight histogram created by AF driver to select best lens position. This histogram corresponds to the situation depicted in Figure 43 and Figure 44. After scanning 10 lens positions numbered 0 through 9, the AF driver determined that Y-normalized sharpness scores from the upper 8 of 16 AF windows (W11 through W24) peak at lens position 0, while the scores from the lower 8 windows (W31 through W44) at lens position 5. For each of the 2 positions, the AF driver summed pre-programmed weights of the AF windows being clearly in focus at that position, thus obtaining 2 position weights. These weights would have been equal if not for very weak dependence of sharpness scores from windows W31, W32, and W33 on lens position. Finding peak-to-trough variability of these scores lower than pre-programmed threshold, the AF driver concluded that for W31, W32, and W33 no lens position was clearly optimal, and therefore the weights of these windows should not be added to the weight of position 5. This gave position 0 a higher weight and decided its selection as the best position. Note: Unequal weighting of the AF windows, increasing the importance to the 4 central ones.

Evaluation of Image Sharpness

Information on image sharpness that the AF algorithm uses to find best focus position is provided by focus measurement engine (FME), a programmable edge-filtering module in Image Flow Processor (IFP). The FME convolves 2 pre-programmed 1-dimensional digital filters with luminance (Y) data that it receives row by row from the color interpolation module. For each interpolated frame, the convolution of the AF filters with Y produces up to 32 local sharpness scores reflecting the density and sharpness of vertical edges in up to 16 user-selected areas of the frame. The FME outputs these sharpness scores once every frame to IFP registers R[77:84]:2.and R[87:94]:2 (where "[:]" denotes a range of register numbers and ":2" means page 2). In addition, the FME calculates and writes to IFP registers R[67:74]:2 up to 16 local averages of Y that can be used to normalize the sharpness scores and thus make them nearly independent of scene brightness.

Since each sharpness score and Y average has only 8 register bits allocated for it, care should be taken in programming the AF filters to ensure that the sharpness scores they produce never exceed 255. Otherwise, the content of registers R[77:84]:2.and R[87:94]:2 may not match actual sharpness scores computed by the FME.

The exact method of computing the sharpness scores is as follows. Sixteen equal size rectangular windows forming a 4 x 4 grid are superimposed on each color-interpolated frame. The size of these AF windows and the position of the upper left corner of the grid are programmable (via IFP registers [R64:66]:2). The grid does not have to be entirely inside the frame. For example, it is perfectly legal to cover most of the frame with a 3x3 portion of the grid and place the remaining 7 AF windows almost entirely outside of it, as shown in Figure 46. Whenever a portion of an AF window is inside the frame, the FME calculates 2 sharpness scores and average Y for this portion. However, it can write these results to registers only if the bottom boundary of the window is partly or fully inside the frame. Placing this boundary entirely outside the frame makes the window inactive, in the sense that the FME stops outputting new sharpness scores and Y averages for it. Although no AF window having some part of the bottom boundary inside the frame can be deactivated in the same sense, any AF window can be made irrelevant in the AF algorithm by giving it a weight of zero.

Each frame is read out from the sensor core and processed by the IFP row by row. Every rectangular block of pixels in the frame can be considered as a separate, smaller frame, also read out and processed row by row. Thinking this way about the AF windows, refer to a portion of a frame row belonging to any AF window as a window row. As the color interpolation module processes each window row and makes Y values of its successive pixels available to the FME, the FME convolves those values with the 2 AF filters. The AF

filters are user-programmable within the following constraints: each can have 8 or 9 integer coefficients with values from -15 to 15, can be symmetric or antisymmetric, and can be multiplied by a power-of-2 weight factor ranging from 1/512 to 32. By default, both are programmed to detect sharp edges, but the first filter is more high-pass than the second. Each filter is applied to successive locations in a window row, starting at the first pixel and ending at the last. This requires using Y values from outside the window, specifically from the 4 columns to the left and 4 columns to the right of the window. Hence, when programming the size and position of the AF window grid, one should make sure that every AF window intended to have non-zero weight is at least than 4 columns away from the left and right side of the frame.

Figure 46: Auto Focus Windows

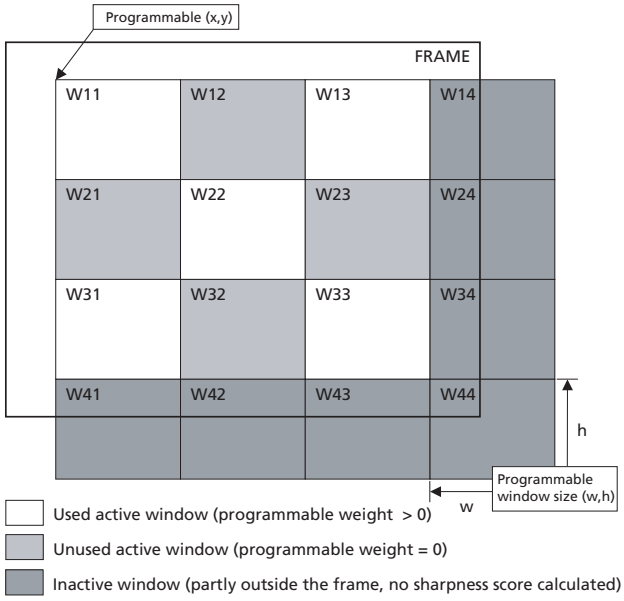


Figure 46 shows an array of 16 equal-size AF windows configured to work like a centered quincunx pattern of 5 windows.

As the convolution of each AF filter with Y progresses along a window row, then to the next row, and so on, absolute values of its successive results are added to a sum that ultimately becomes a sum over the whole portion of the window located inside the frame. At the same time, the pixels in the window are counted and their Y values are added up to get the average Y for the window.

In this way, schematically depicted in Figure 47, each AF window not located fully outside the frame yields 2 sharpness scores (the sums of convolution results from the 2 AF filters) and 1 average Y. The number of window rows processed to obtain these results can be equal to or less than the common AF window height programmed into the register R65:2. If and only if the window row count matches that height, the results are output to registers. This never happens for AF windows positioned like W41 or W44 in Figure 46—hence these windows are inactive. Results from each active AF window are output immediately after its last row is processed.

Figure 47: Computation of Sharpness Scores and Luminance Average for an AF Window

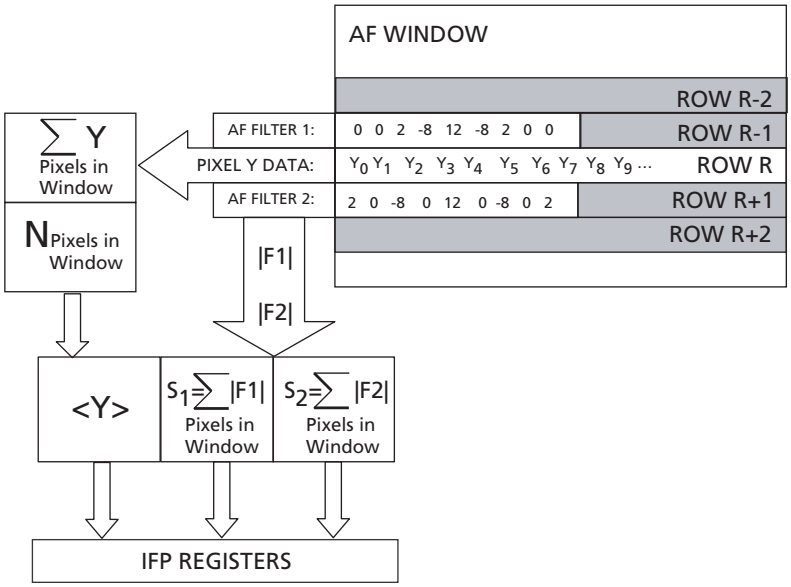


Figure 47 shows the computation of sharpness scores and average luminance in an AF window. Coefficients of the two AF filters are programmable. The filters shown here as an example yield convolution results $F_1 = 2Y_2 - 8Y_3 + 12Y_4 - 8Y_5 + 2Y_6$ and $F_2 = 2Y_0 - 8Y_2 + 12Y_4 - 8Y_6 + 2Y_8$.

The symmetry constraint placed on the AF filters reduces the number of coefficient values needed to define them. Symmetric 8- or 9-coefficient filters are defined by specifying 4 or 5 coefficient values, respectively. Only 4 coefficients are needed to define an 8- or 9-coefficient antisymmetric filter. Examples of AF filters that can be programmed into MT9D111 are given in Table 41. To program the first AF filter, one must write its parameters to IFP registers R75:2 and R76:2. The parameters of the second AF filter must be written to IFP registers R85:2 and R86:2.

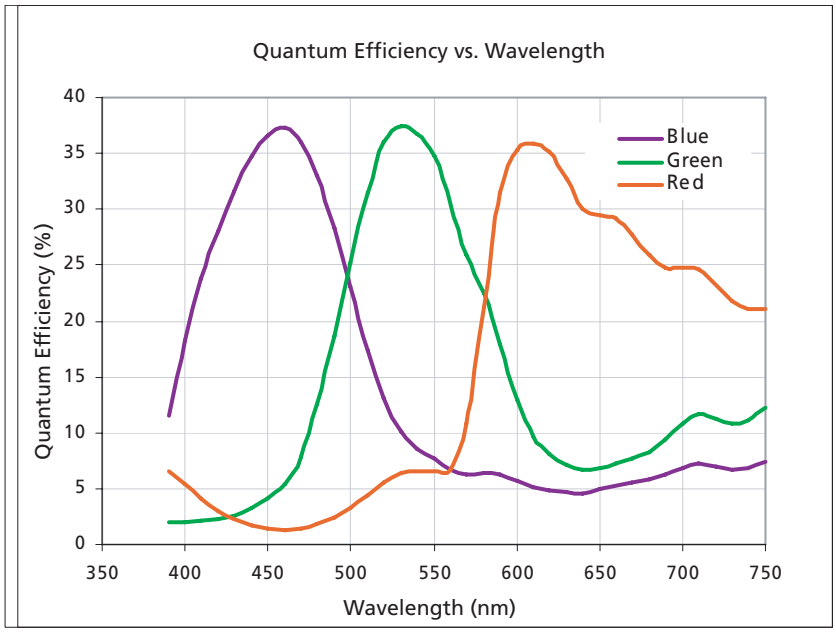
.

Table 41: Examples of AF Filters that can be Programmed into the MT9D111

Filter Parameters Programmed into Registers								Filter
Filter Size	Filter Symmetry	Filter Coefficients					Filter Weight	
		C0	C1	C2	C3	C4		
8	symmetric	n/a	6	-7	2	0	1	[0 2 -7 6 6 -7 2 0]
8	antisymmetric	n/a	1	0	0	0	1/4	[0 0 0 -1/4 1/4 0 0 0]
9	symmetric	6	0	-4	0	1	2	[2 0 -8 0 12 0 -8 0 2]
9	antisymmetric	0	15	5	0	0	1/8	[0 0 -5/8 -15/8 0 15/8 5/8 0 0]

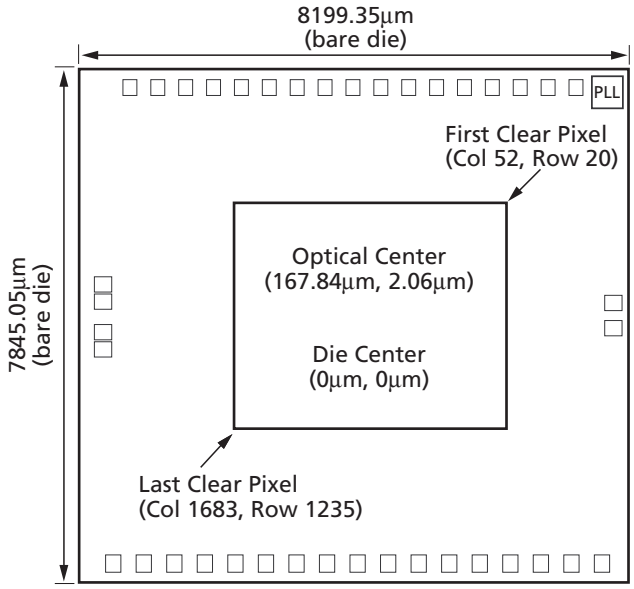
Spectral Characteristics

Figure 48: Typical Spectral Characteristics



Die Outline

Figure 49: Optical Center Offset



Note: Figure not to scale.

Electrical Specifications

Recommended die operating temperature range is from -20° to +55°C. The sensor image quality may degrade above +55°C.

Table 42: AC Electrical Characteristics

Symbol	Definition	Conditions	MIN	TYP	MAX	Units
f_{CLKIN1}	Input clock frequency	PLL enabled (MCLK max = 80 MHz)	6	10	64	MHz
t_{CLKIN1}	Input clock period	PLL enabled (MCLK max = 80 MHz)	15.625	100	166.7	ns
f_{CLKIN2}	Input clock frequency	PLL disabled	6		80	MHz
t_{CLKIN2}	Input clock period	PLL disabled	12.5		166.7	ns
t_R	Input clock rise time		0.5		1	V/ns
t_F	Input clock fall time		0.5		1	V/ns
	Clock duty cycle		40	50	60	%
f_{PIXCLK}	PIXCLK frequency	Default				MHz
t_{PD}	PIXCLK to data valid	Default	-3		3	ns
t_{PFH}	PIXCLK to FV high	Default	-3		3	ns
t_{PLH}	PIXCLK to LV high	Default	-3		3	ns
t_{PFL}	PIXCLK to FV low	Default	-3		3	ns
t_{PLL}	PIXCLK to LV low	Default	-3		3	ns
C_{IN}	Input pin capacitance			3.5		pF
C_{LOAD}	Load capacitance			15	20	pF
AC Setup Conditions						
	f_{CLKIN1}		6		64	MHz
	V_{DD}		1.7	1.8	1.95	V
	$V_{DD}Q$		1.7	2.8	3.1	V
	V_{AA}		2.5	2.8	3.1	V
	VAAPIX		2.5	2.8	3.1	V
	V_{DD}PLL		2.5	2.8	3.1	V
	Output load			15		

Table 43: DC Electrical Definitions and Characteristics

Symbol	Definition	Conditions	MIN	TYP	MAX	Units	Note
VDD	Core digital voltage		1.7	1.8	1.95	V	
VDDQ	I/O digital voltage		1.7	2.8	3.1	V	
VDDGPIO	GPI/O digital voltage		1.7	2.8	3.1	V	
VAA	Analog voltage		2.5	2.8	3.1	V	
VAAPIX	Pixel supply voltage		2.5	2.8	3.1	V	
VDDPLL	PLL supply voltage		2.5	2.8	3.1	V	
VIH	Input high voltage	VDDQ = 2.8V	2.4		VDDQ+0.3	V	
		VDDQ = 1.8V	1.4		VDDQ+0.3		
VIL	Input low voltage	VDDQ = 2.8V	GND-0.3		0.8	V	
		VDDQ = 1.8V	GND-0.3		0.5		
IIN	Input leakage current	No pull-up resistor; VIN = VDDQ or DGND	-10	+/-0.5	10	µA	
VOH	Output high voltage	At specified IOH	VDDQ-0.4			V	
VOL	Output low voltage	At specified IOL			0.4	V	
IOH	Output high current	At specified VOH = VDDQ~400MV AT 1.7V VDDQ	-7		x	mA	
IOL	Output low current	At specified VOL~400MV at 1.7V VDDQ	7		x	mA	
IOZ	Tri-state output leakage current	Vin =VDDQ or GND	-10	+/-0.5	10	µA	
IDD1	Digital operating current	Context B, 1600x1200, JPEG on, MCLK=Max, PIXCLK=Max		92	110	mA	
IDDQ1	I/O digital operating current	Context B, 1600x1200, JPEG on, MCLK=Max, PIXCLK=Max		1.5		mA	1
IAA1	Analog operating current	Context B, 1600x1200, JPEG on, MCLK=Max, PIXCLK=Max		43	55	mA	
IAAPIX1	Pixel supply current	Context B, 1600x1200, JPEG on, MCLK=Max, PIXCLK=Max		2.1	3.5	mA	
IDDPLL1	PLL supply current	Context B, 1600x1200, JPEG on, MCLK=Max, PIXCLK=Max		2.3	3	mA	
IDD2	Digital operating current	Context A, 800x600, No JPEG, MCLK=Max, PIXCLK=Max		48	60	mA	
IDDQ2	I/O digital operating current	Context A, 800x600, No JPEG, MCLK=Max, PIXCLK=Max		15		mA	1
IAA2	Analog operating current	Context A, 800x600, No JPEG, MCLK=Max, PIXCLK=Max		27	35	mA	
IAAPIX2	Pixel supply current	Context A, 800x600, No JPEG, MCLK=Max, PIXCLK=Max		4	5.5	mA	
IDDPLL2	PLL supply current	Context A, 800x600, No JPEG, MCLK=Max, PIXCLK=Max		2.3	3	mA	
ISTDBY1	Standby current PLL enabled	PLL enabled (MCLK = 0Hz, held at either VIL or VIH)		35	100	µA	
ISTDBY2	Standby current PLL disabled	PLL disabled (MCLK = 0Hz, held at VIL or VIH)		35	100	µA	

Notes: 1. Due to the influence of several variables (scene illumination, output load) max values are not available.

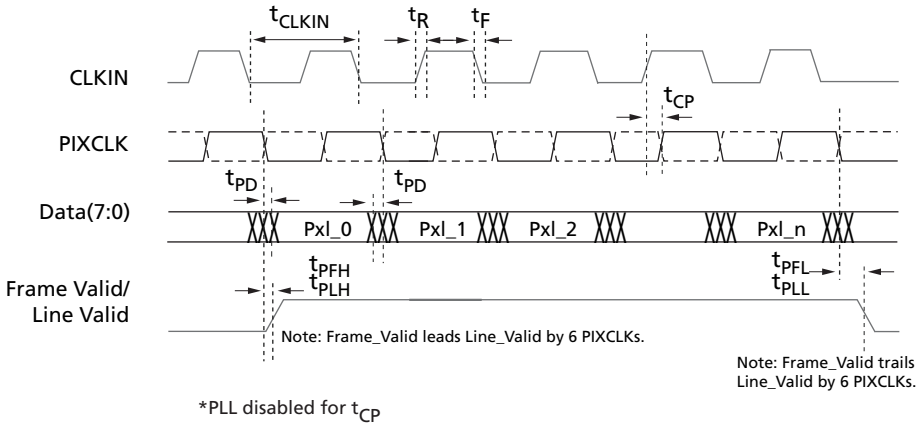
Table 44: Absolute Maximum Ratings

Symbol	Parameter	MIN	MAX	Unit
V_{DD}	Digital power	-0.3	2.4	V
V_{DDQ}	I/O power	-0.3	4.0	V
V_{DDPLL}	PLL power	-0.3	4.0	V
V_{AA}	Analog power (2.8V)	-0.3	4.0	V
VAAPIX	Pixel array power	-0.3	4.0	V
V_{IN}	DC input voltage	-0.3	V_{DDQ}+0.3	V
V_{OUT}	DC output voltage	-0.3	V_{DDQ}+0.3	V
T_{OP}	Operation temperature	-30	70	°C
T_{STG}[1]	Storage temperature	-40	85	°C

Note: [1]Stresses above those listed may cause permanent damage to the device. This is a stress rating only, and functional operation of the device at these or any other conditions above those indicated in the product specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

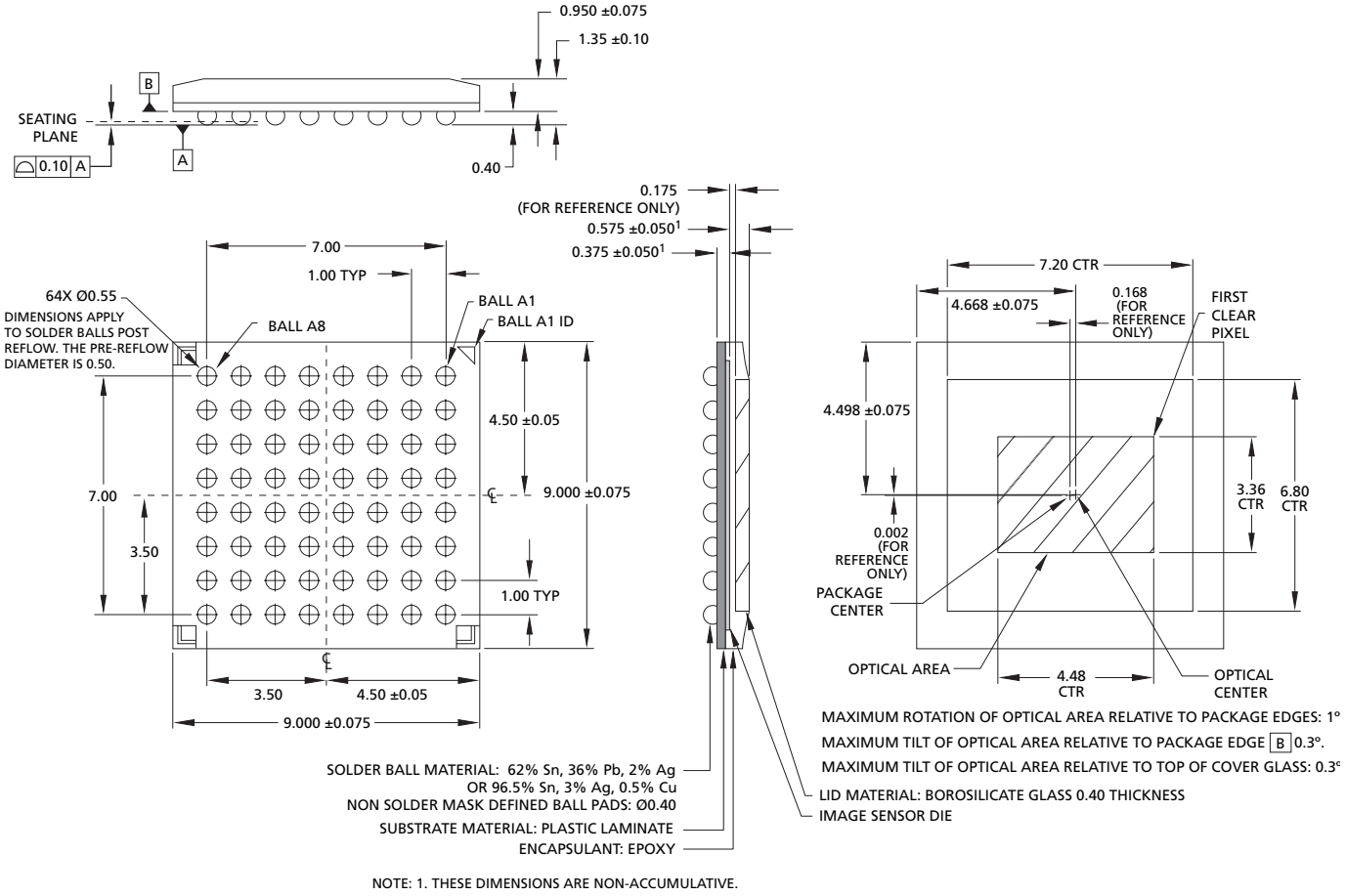
I/O Timing

Figure 50: I/O Timing Diagram



Packaging

Figure 51: 64-Ball iCSP Package Mechanical Drawing



NOTE: 1. THESE DIMENSIONS ARE NON-ACCUMULATIVE.

All dimensions in millimeters.

8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, Tel: 208-368-3900
prodmktg@micron.com www.micron.com Customer Comment Line: 800-932-4992

Appendix A: Two-Wire Serial Register Interface

This section describes the two-wire serial interface bus that can be used in any functional sensor mode.

The two-wire serial interface bus enables R/W access to control and status registers within the sensor core.

The interface protocol uses a master/slave model in which a master controls one or more slave devices. The sensor acts as a slave device. The master generates a clock (SCLK) that is an input to the sensor and used to synchronize transfers. The master is responsible for driving a valid logic level on SCLK at all times. Data is transferred between the master and the slave on a bidirectional signal (SDATA). Both the SDATA AND SCLK signal are pulled up to VDD off-chip by a 1.5KΩ resistor. Either the slave or master device can drive the SDATA line low—the interface protocol determines which device is allowed to drive the SDATA line at any given time.

Protocol

The two-wire serial interface bus defines the transmission codes as follows:

- a start bit
- the slave device 8-bit address
- a(an) (no) acknowledge bit
- an 8-bit message
- a stop bit

Sequence

A typical read or write sequence is executed as follows:

1. The master sends a start bit.
2. The master sends the 8-bit slave device address. The last bit of the address determines if the request is a read or a write, where a “0” indicates a write and a “1” indicates a read.
3. The slave device acknowledges receipt of the address by sending an acknowledge bit to the master.
4. If the request is a write, the master then transfers the 8-bit register address, indicating where the write takes place.
5. The slave sends an acknowledge bit, indicating that the register address has been received.
6. The master then transfers the data, 8 bits at a time, with the slave sending an acknowledge bit after each 8 bits.

The sensor core uses 16-bit data for its internal registers, thus requiring two 8-bit transfers to write to one register. After 16 bits are transferred, the register address is automatically incremented so that the next 16 bits are written to the next register address. The master stops writing by sending a start or stop bit.

A typical read sequence is executed as follows.

1. The master sends the write-mode slave address and 8-bit register address, just as in the write request.
2. The master then sends a start bit and the read-mode slave address, and clocks out the register data, 8 bits at a time.
3. The master sends an acknowledge bit after each 8-bit transfer. The register address is auto-incremented after every 16 bits is transferred.
4. The data transfer is stopped when the master sends a no-acknowledge bit.

Bus Idle State

The bus is idle when both the data and clock lines are high. Control of the bus is initiated with a start bit, and the bus is released with a stop bit. Only the master can generate start and stop bits.

Start Bit

The start bit is defined as a HIGH-to-LOW data line transition while the clock line is HIGH.

Stop Bit

The stop bit is defined as a LOW-to-HIGH data line transition while the clock line is HIGH.

Slave Address

The 8-bit address of a two-wire serial interface device consists of seven bits of address and one bit of direction. A "0" in the LSB (least significant bit) of the address indicates write mode, and a "1" indicates read mode. The default slave addresses used by the sensor core are 0xBA (write address) and 0xBB (read address). R0x0D:0[10] or the SADDR pin can be used to select the alternate slave addresses 0x90 (write address) and 0x91 (read address).

Writes to R0x0D:0[10] are inhibited when the standby pin is asserted (all other writes proceed normally). This allows two sensors to co-exist as slaves on this interface, but they must be addressed independently. Enable this capability as follows:

After RESET, both sensors use the default slave address. Reads or writes on the serial register interface to the default slave address are decoded by both sensors simultaneously.

1. After RESET, assert the STANDBY signal to one sensor and negate the STANDBY signal to the other sensor.
2. Perform a write to R0x0D:0 with bit 10 set. The sensor with STANDBY asserted ignores the write to bit 10 and continues to decode at the default slave address.

The sensor with STANDBY negated has its R0x0D:0[10] set and responds to the alternate slave address for all subsequent READ and WRITE operations, as shown in Table 45.

Table 45: Slave Address Options

ADDR	R0xD:0[10]	Slave Address	
		WRITE	Read
0	0	0x090	0x091
0	1	0x0BA	0x0BB
1	0	0x0BA	0x0BB
1	1	0x090	0x091

Data Bit Transfer

One data bit is transferred during each clock pulse. The serial interface clock pulse is provided by the master. The data must be stable during the high period of the two-wire serial interface clock—it can only change when the serial clock is low. Data is transferred eight bits at a time, followed by an acknowledge bit.

Acknowledge Bit

The master generates the acknowledge clock pulse. The transmitter (which is the master when writing, or the slave when reading) releases the data line, and the receiver indicates an acknowledge bit by pulling the data line low during the acknowledge clock pulse.

No-Acknowledge Bit

The no-acknowledge bit is generated when the data line is not pulled down by the receiver during the acknowledge clock pulse. A no-acknowledge bit is used to terminate a read sequence.

Page Register

The MT9D111 two-wire serial interface and its associated protocols support an address space of 256 16-bit locations. This address space is extended by a 3-bit page prefix, and controlled through accesses to R0xF0:0.

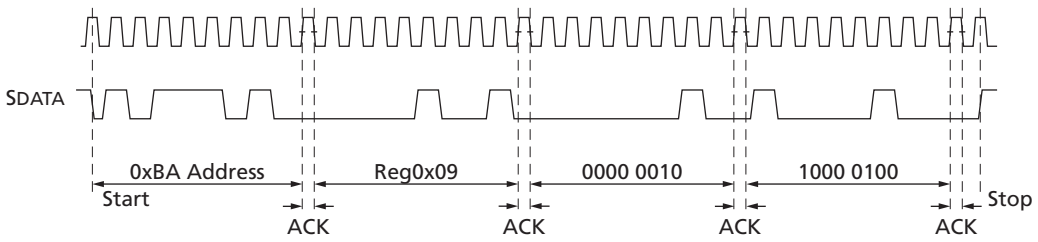
The paging mechanism is intended to allow access to other sets of registers when the sensor is embedded as part of a more complex integrated subsystem, for example, in an SOC. All registers within the sensor core are accessible on page 0 (the default page).

Sample Write and Read Sequences

16-Bit Write Sequence

A typical write sequence for writing 16 bits to a register is shown in Figure 52. A start bit given by the master starts the sequence, followed by the write address. The image sensor then sends an acknowledge bit and expects the register address to come first, followed by the 16-bit data. After each 8-bit transfer, the image sensor sends an acknowledge bit. All 16 bits must be written before the register is updated. After 16 bits are transferred, the register address is automatically incremented so that the next 16 bits are written to the next register. The master stops writing by sending a start or stop bit.

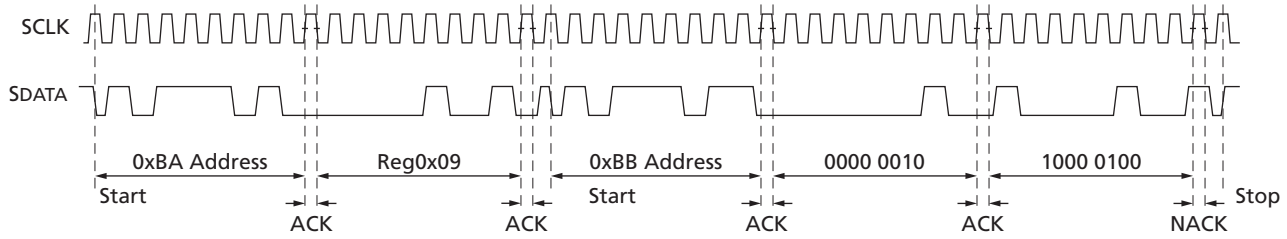
Figure 52: WRITE Timing to R0x09:0—Value 0x0284



16-Bit Read Sequence

A typical read sequence is shown in Figure 53. First the master writes the register address, as in a write sequence. Then a start bit and the read address specify that a read is about to happen from the register. The master clocks out the register data, eight bits at a time. The master sends an acknowledge bit after each 8-bit transfer. The register address should be incremented after every 16 bits is transferred. The data transfer is stopped when the master sends a no-acknowledge bit.

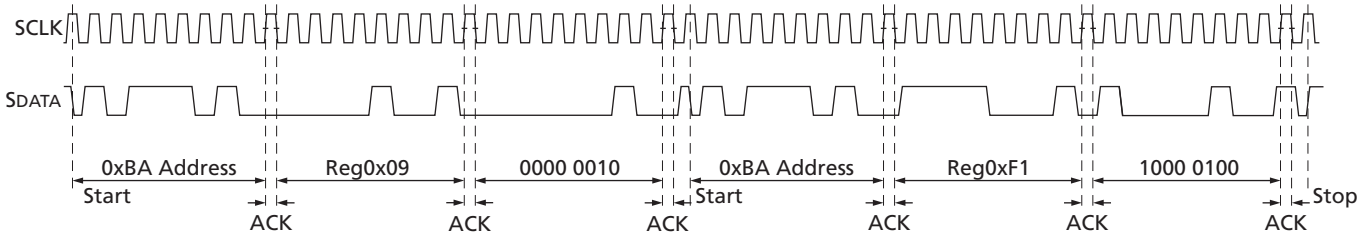
Figure 53: READ Timing from R0x09:0; Returned Value 0x0284



8-Bit Write Sequence

To be able to write one byte at a time to the register, a special register address is added. The 8-bit write is done by writing the upper 8 bits to the desired register, then writing the lower 8 bits to the special register address (R0xF1:0). The register is not updated until all 16 bits have been written. It is not possible to update just half of a register. In Figure 54, a typical sequence for 8-bit writes is shown. The second byte is written to the special register (R0xF1:0).

Figure 54: WRITE Timing to R0x09:0—Value 0x0284



8-Bit Read Sequence

To read one byte at a time, the same special register address is used for the lower byte. The upper 8 bits are read from the desired register. By following this with a read from the special register (R0xF1:0), the lower 8 bits are accessed (Figure 55). The master sets the no-acknowledge bits.

Figure 55: READ Timing from R0x09:0; Returned Value 0x0284

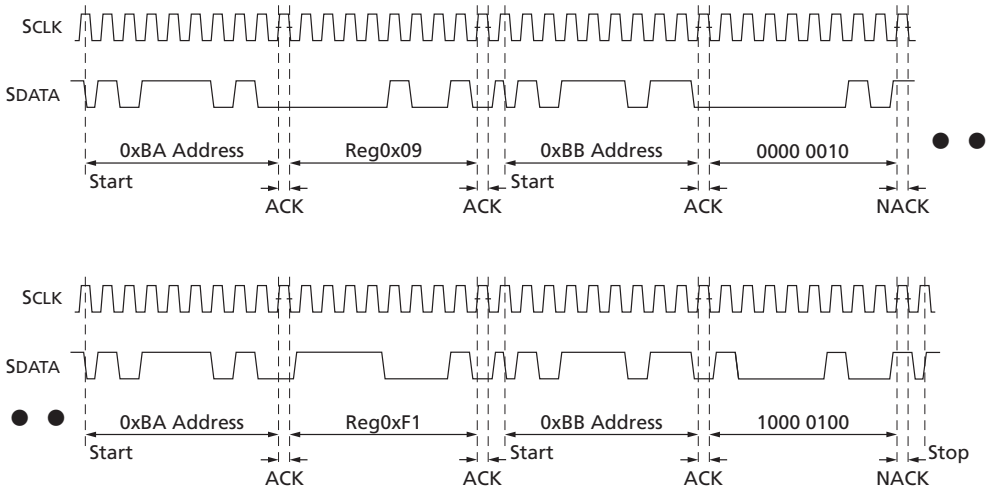
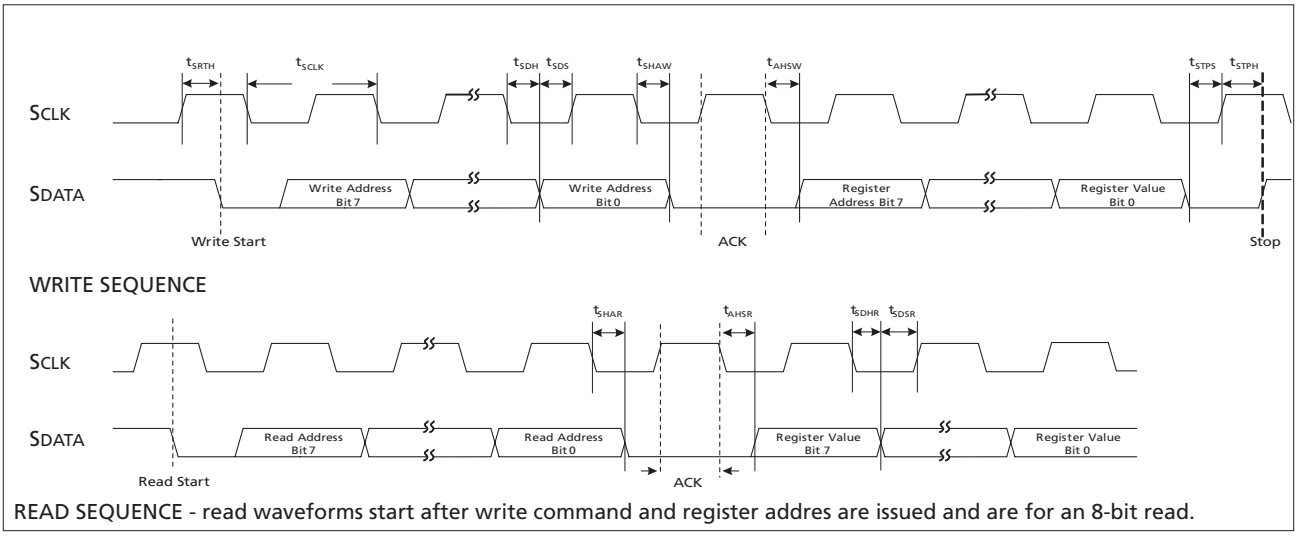


Figure 56: Two-Wire Serial Bus Timing Parameters



READ SEQUENCE - read waveforms start after write command and register addres are issued and are for an 8-bit read.

Table 46: Two-wire Serial Bus Characteristics

Symbol	Definition	Conditions	MIN	TYP	MAX	Units
^{f}SCLK	Serial interface input clock frequency				^{f}CLK/16	kHz
^{t}SCLK	Serial interface input clock period		1/^{t}SCLK			ns
	SCLK duty cycle		40	50	60	%
^{t}SRTH	Start hold time	WRITE/READ	4*^{t}CLK			ns
^{t}SDH	SDATA hold	WRITE	4*^{t}CLK			ns
^{t}SDS	SDATA setup	WRITE	4*^{t}CLK			ns
^{t}SHAW	SDATA hold to ACK	WRITE	4*^{t}CLK			ns
^{t}AHSW	ACK hold to SDATA	WRITE	4*^{t}CLK			ns
^{t}STPS	Stop setup time	WRITE/READ	4*^{t}CLK			ns
^{t}STPH	Stop hold time	WRITE/READ	4*^{t}CLK			ns
^{t}SHAR	SDATA hold to ACK	READ	4*^{t}CLK			ns
^{t}AHSR	ACK hold to SDATA	READ	4*^{t}CLK			ns
^{t}SDHR	SDATA hold	READ	4*^{t}CLK			ns
^{t}SDSR	SDATA setup	READ	4*^{t}CLK			ns
CIN_SI	Serial interface input pin capacitance			3.5		pF
CLOAD_SD	SDATA max load capacitance			15		pF
RSD	SDATA pull-up resistor			1.5		KΩ

Note: Either the slave or master device can drive the SCLK line low—the interface protocol determines which device is allowed to drive the SCLK line at any given time.

Revision History

Rev. B, Production 1/06

- Add Table 2, "Available Part Numbers," on page 1
- Update Table 3, "Signal Description," on page 12
- Update Table 4, "Sensor Core Register Defaults," on page 27
- Update Table 10, "Driver Variables-Monitor Driver (ID = 0)," on page 69
- Update Table 11, "Driver Variables-Sequencer Driver (ID = 1)," on page 70
- UpdateTable 13, "Driver Variables-Auto White Balance (ID = 3)," on page 77
- Update Table 15, "Driver Variables-Auto Focus Driver (ID = 5)," on page 82
- Update Table 16, "Driver Variables-Auto Focus Mechanics Driver (ID = 6)," on page 85
- Update "Changes to Gain Settings" on page 122
- Update"Window Size" on page 124
- Update "Column and Row Skip" on page 125
- Update "Context Switching" on page 129
- Update "Two-Wire Serial Interface Programming" on page 138
- Update "Auto Focus Mechanics Driver" on page 145
- Update "Flicker Avoidance Driver" on page 146
- Update "Power-Up" on page 149
- Update "Hard Reset Sequence" on page 149
- Add Figure 33, Power On/Off Sequence, on page 150
- Update "Enable PLL" on page 150
- Update "Configure Pad Slew" on page 151
- Update "Standby Sequence" on page 151
- Update "To Exit Standby" on page 152
- Add Figure 34, Hard Standby Sequence, on page 153
- Update Figure 48, Typical Spectral Characteristics, on page 175
- Update Table 46, "Two-wire Serial Bus Characteristics," on page 185

Rev. A, Advance 7/05

- Initial release