AS0147AT Host Command Interface Specification

AND9929/D

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE

Introduction

This document describes the Host Command Interface between the AS0147AT and a Host microcontroller. This document is intended for system developers to write Host controller firmware, and for flash memory content providers.

TABLE OF CONTENTS

OVERVIEW

BACKGROUND

The AS0147AT is an automotive/surveillance sensor plus companion−chip, building on the capabilities of its predecessor: AP0100. The high−level host command interface has been maintained and extended for AS0147AT, and in many cases it is backwards−compatible with AP0100. New commands have been added to support new features. Full details of the changes introduced for AS0147AT are provided in “Appendix H: Changes Since AP0100 Rev 2” on page 120.

Host Command Interface Context

Figure 1 shows the hardware context of the host command interface for the AS0147AT. In this configuration, the host controls the AS0147AT through its camera control interface (CCI). CCI is the standard two−wire serial interface. This gives the Host access both to the internal RAM of the device, and to its peripherals. However, the Host interface interacts with the device through its Command Register.

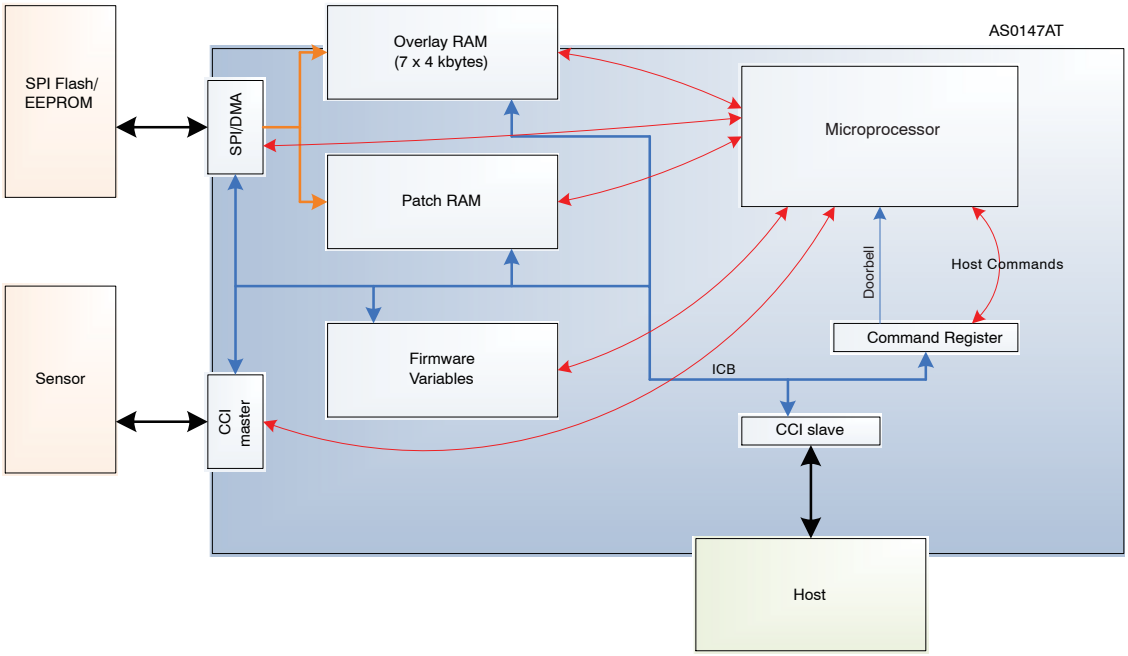

Figure 1. Host Command Interface Context via CCI

ARCHITECTURAL CONCEPTS

The AS0147AT hardware implements a 16−bit command register. Bit 15 of this register is a ‘sticky’ doorbell interrupt to the AS0147AT microprocessor. The bit is ’sticky’ in that the Host can set the bit, but it cannot clear it; only the AS0147AT firmware can clear the bit.

The AS0147AT firmware provides the Command Handler task, which is responsible for processing Host commands, and returning the command status (and any optional response parameters) to the Host. The firmware also provides a logical command handler variable page. This page contains the Parameters Pool, a region of RAM to contain command parameters and response parameters. The size of this region of RAM determines the maximum size of any command’s parameters or response parameters.

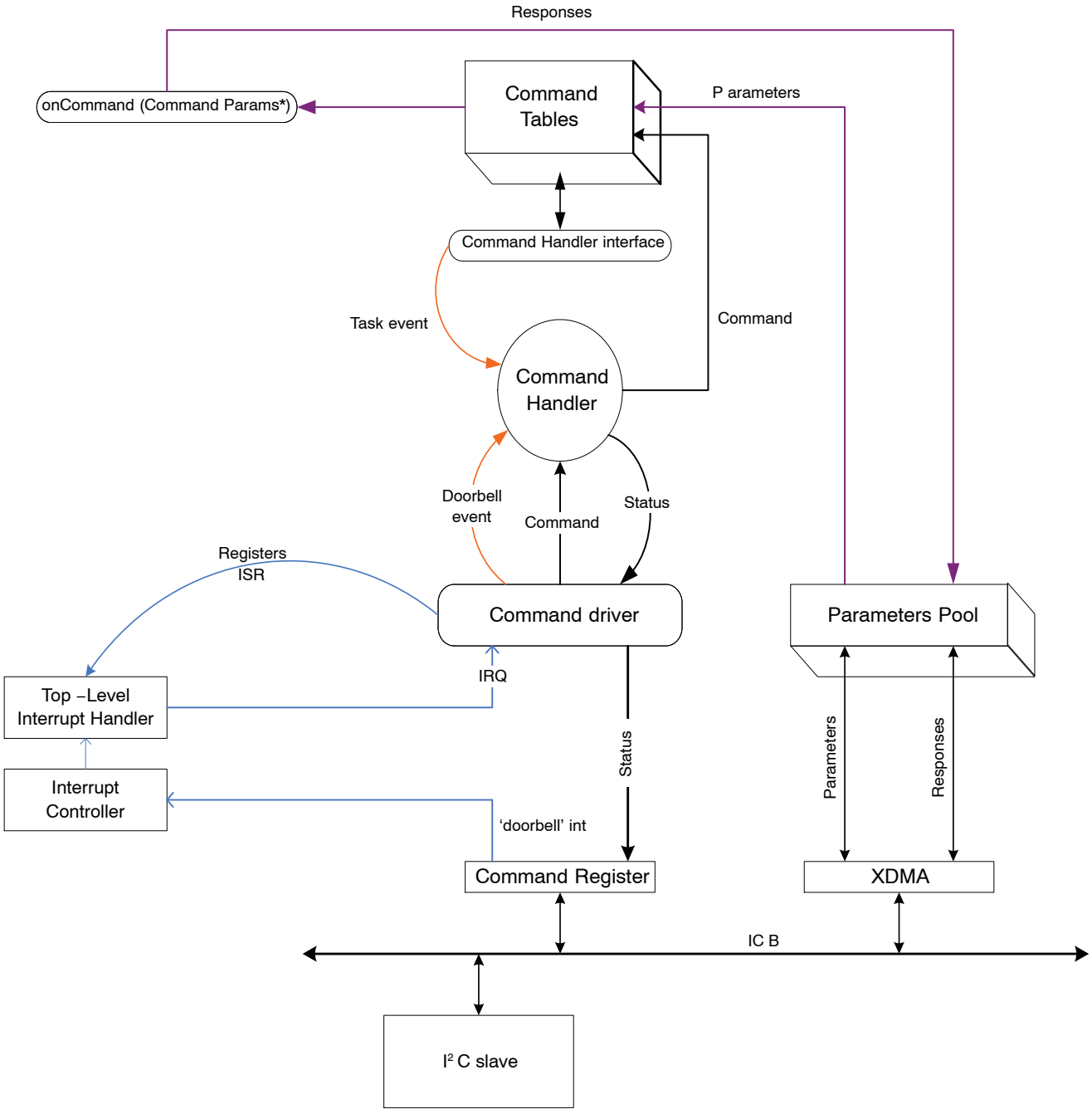

Figure 2. Command Processing Components

Internally, the Command Handler provides 'Command Tables'. These contain 'handlers' for each command supported by the firmware. When a host issues a command, the Command Handler task searches the Command Tables to locate a handler, which it then invokes. Each handler is invoked with an abstract pointer to the Parameters Pool, to allow each command access to its parameters.

Command handlers can be either synchronous, which means they execute completely within the context of the command handler task, or asynchronous, which means the command handler signals a separate task to process the command. In the synchronous case, the return value from the handler function represents the final state of the command. In the asynchronous case, the return value from the handler only represents the fact that the command was valid and accepted, and is in progress.

The command handler always waits for the command handler function to complete. Then it writes the returned command result status to the command register. This clears the doorbell bit, indicating to the Host that the command has completed. The Host is now free to retrieve any response parameters and to issue another command.

HOST COMMAND PROCESSING

The Host command processing sequence is shown in Figure 3.

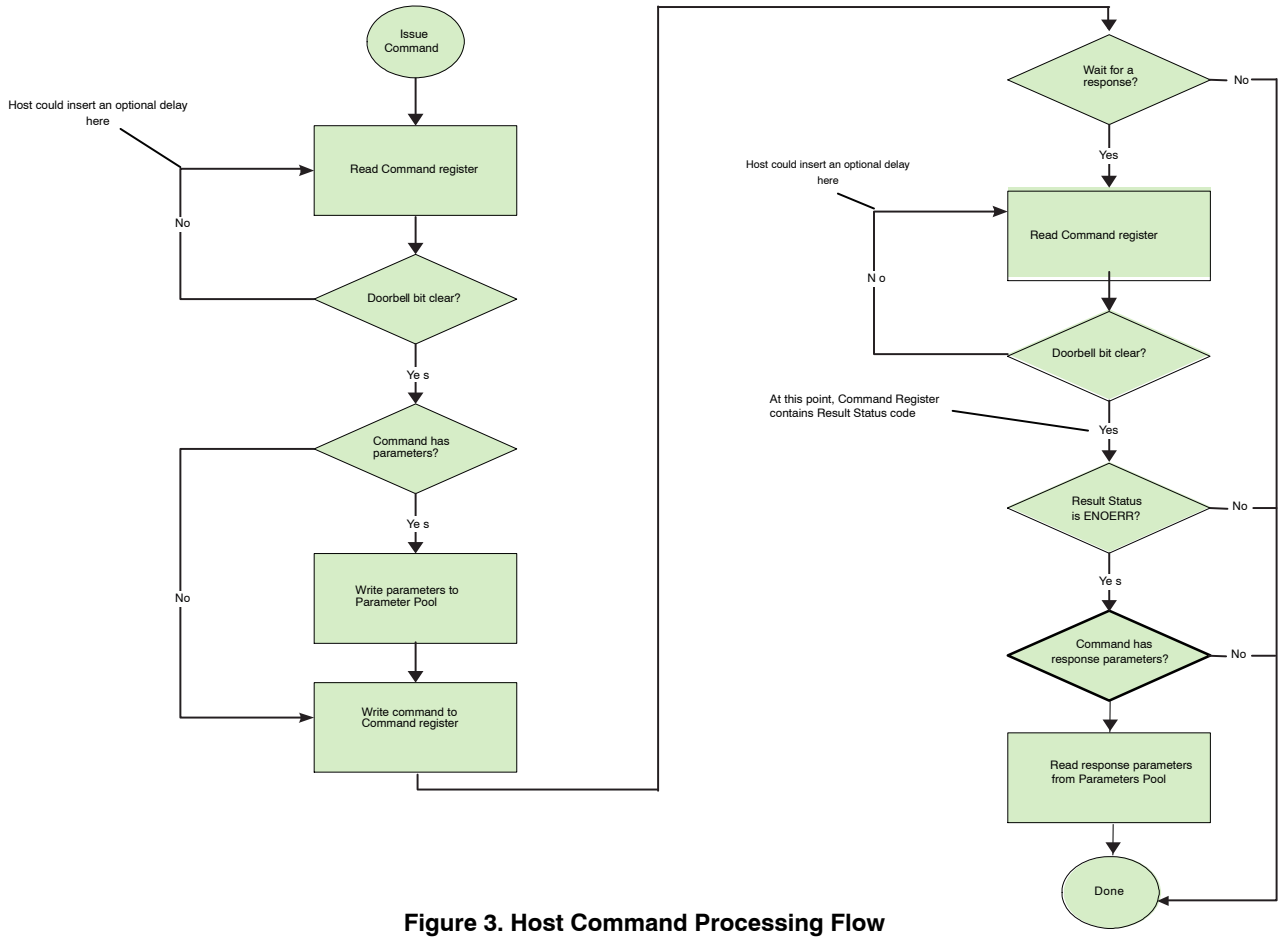

Figure 3. Host Command Processing Flow

The Host issues a command by writing (through the CCI) to the Command Register. All commands are encoded with bit 15 set, which automatically generates the Host command (doorbell) interrupt to the microprocessor.

Assuming initial conditions, the Host first writes the command parameters (if any) to the parameters pool (in the command handler's shared−variable page), then writes the command to the Command Register. The firmware's interrupt handler is invoked, which immediately copies the Command Register contents (to mitigate against accidental overwriting by the host). The interrupt handler then signals the Command Handler task to process the command (as described in "Architectural Concepts" on page 4).

If the Host wishes to determine the outcome of the command, it must poll the Command Register waiting for the doorbell bit to be cleared. This indicates that the firmware completed processing the command. The contents of the Command Register indicate the command's result status. If the command generated response parameters, the Host can now retrieve these from the parameters pool.

Note: The Host must not write to the parameters pool, nor issue another command, until the previous command

completes. This is true even if the Host does not care about the result of the previous command. It is strongly recommended that the Host tests that the doorbell bit is clear before issuing a command.

Example C source code illustrating how a host can issue a 'Change−Config' request via the CCI interface is provided in "Appendix F: Host Command Usage Example" on page 111.

Synchronous Command Flow

The typical flow for synchronous commands is:

1. The Host issues a command to perform an operation.
2. The registered command handler is invoked, validates the command parameters, and then performs the operation. The handler returns the command result status to indicate the result of the operation.
3. The Host retrieves the command result value, and any associated command response parameters.

Asynchronous Command Flow

The typical flow for asynchronous commands is:

1. The Host issues a command to start an operation.
2. The registered command handler is invoked, validates and copies the command parameters, then signals a separate task to perform the operation. The handler returns the ENOERR return value to indicate the command was acceptable and is in progress.
3. The Host retrieves the command return value – if it is not ENOERR (see Table 3, "Response Codes," on page 8) the Host knows that the command was not accepted and is not in progress.
4. Subsequently, the Host issues an appropriate get status command to both poll whether the command has completed, and if so, retrieve any associated response parameters.
5. The registered command handler is invoked, determines the state of the command (via shared variables with the processing task), and returns either EBUSY to indicate the command is still in progress, or it returns the result status of the command.
6. The Host must re−issue the get status command until it does not receive the EBUSY (see Table 3) response.

Asynchronous commands exist to allow the Host to issue multiple commands to the various subsystems without having to wait for each command to complete. This prevents the Host command interface from being blocked by a long−running command. Therefore, each asynchronous command has a get status (or similar) command to allow the Host to determine when the asynchronous command has completed (see Table 1).

Table 1. ASYNCHRONOUS COMMANDS AND THEIR 'GET STATUS' PARTNER

Component	Asynchronous Command	'Get Status' Command
Overlay Manager	Load Buffer	Load Status
	Load String0	Load Status
	Load String1	Load Status
	Load String2	Load Status
	Load String3	Load Status
STE Manager	Load Config	Load Status
Flash Manager	Get Lock	Lock Status
	Read	Flash Status
	Write	Flash Status
	Erase Block	Flash Status
	Erase Device	Flash Status
	Query Device	Flash Status
Patch Loader	Load Patch	Status
	Apply Patch	Status
Sequencer	Refresh	Refresh Status
Calib Stats	Control	Read
CCI Manager	Get Lock	Lock Status
	Read	CCI Status
	Write	CCI Status
	Write Bit−field	CCI Status

Each asynchronous command will return a result status almost immediately to indicate whether the command was accepted. ENOERR means that the command was valid, and is now in progress. It is possible for an asynchronous command to not be accepted, in which case something other than ENOERR will be returned. This indicates that the command is not in progress. Note that in all cases, the doorbell bit has to go to zero before the Host can interpret the result status code.

START−UP HOST COMMAND LOCKOUT

The AS0147AT firmware implements an internal Host Command lock. At start−up, the firmware obtains this lock, which prevents the Host from successfully issuing a Host command. All Host commands will be rejected with EBUSY until the lock is freed.

The firmware releases the Host Command lock when it completes its start−up processing. The time to do this is dependent upon the configuration mechanism. It is recommended that the Host poll the device with the System Manager Get State command until ENOERR is returned.

MULTITASKING

The AS0147AT firmware is multitasking. Therefore it is possible for an internally requested command to be in progress when the Host issues a command. In these circumstances, the Host command is immediately rejected with EBUSY. The Host should reissue the command after a short interval.

COMMAND SEQUENCE PROCESSING

The AS0147AT firmware supports command sequence records stored in SPI non−volatile memory (NVM). A command sequence consists of one or more register/variable updates and/or Host commands. A command sequence can only contain a subset of commands as detailed in “Appendix F: Host Command Usage Example” on page 111.

HOST COMMANDS

OVERVIEW

The AS0147AT supports a number of functional modules or processing subsystems. Each module or subsystem exposes commands to the Host to control and configures its operation. The Host Interface is extensible – commands are not hard−coded. This permits firmware patches to be loaded that offer new (or extended) commands (through the command handler table).

COMMAND PARAMETERS

Command parameters are written to the Parameters Pool shared variables by the host prior to invoking the command. Similarly, any Command Response parameters are also written back to the Parameters Pool by the firmware.

Base Parameter Types

All command parameters use (or are built from) the supported set of base data types as detailed in Table 2.

Table 2. BASE PARAMETER TYPES

Base Type	Size	Description
UINT8	8 bits	Unsigned integer (0...255)
INT8	8 bits	Signed integer (−128...127)
UINT16	16 bits	Unsigned integer (0...65535)
INT16	16 bits	Signed integer (−32768...32767)
UINT32	32 bits	Unsigned integer (0... 4294967295)
INT32	32 bits	Signed integer (−2147483648 ... 2147483647)
FLAG	8 bits	Boolean (0 = FALSE, non−zero = TRUE)
BITFIELD_8	8 bits	Set of bits − encoding specific to parameter
BITFIELD_16	16 bits	Set of bits − encoding specific to parameter
BITFIELD_32	32 bits	Set of bits − encoding specific to parameter
FLOAT	32−bit	IEEE 754 floating point

NOTE: All multibyte types are encoded in big−endian format; see “Appendix A: Big−Endian Encoding” on page 102 for details.

RESULT STATUS CODES

Table 3 shows the result status codes that are written by the command handler to the Host command register, in response to a command.

Table 3. RESPONSE CODES

Value	Mnemonic	Typical Interpretation
0x00	ENOERR	No error – command was successful
0x01	ENOENT	No such entity
0x02	EINTR	Operation interrupted
0x03	EIO	I/O failure
0x04	E2BIG	Too big
0x05	EBADF	Bad file/handle
0x06	EAGAIN	Would−block, try again
0x07	ENOMEM	Not enough memory/resource
0x08	EACCES	Permission denied
0x09	EBUSY	Entity busy, cannot support operation
0x0A	EEXIST	Entity exists
0x0B	ENODEV	Device not found
0x0C	EINVAL	Invalid argument
0x0D	ENOSPC	No space/resource to complete
0x0E	ERANGE	Parameter out of range
0x0F	ENOSYS	Operation not supported
0x10	EALREADY	Already requested/exists

NOTE: Any unrecognized host commands will be immediately rejected by the Command Handler, with result status code ENOSYS.

IMPLEMENTATION CONSTANTS

Each implementation of the Host command interface imposes limits upon resources; for example, address ranges or available buffer sizes. Table 4 lists constants used throughout this document to refer to these implementation−dependent limits, and details the implementation of these constants.

Table 4. IMPLEMENTATION CONSTANTS

Constant	Implementation	Description
PARAMS_POOL_SIZE	256 bytes	Size of the Parameters Pool (in bytes)
GPIO_MAX_GPI_ASSOCIATIONS	8	Maximum number of GP input associations supported by the GPIO Manager
GPIO_GPI_SAMPLE_PERIOD	33 ms	Period between GPI samples (for association processing)
CMDHANDLER_MAX_CMDSEQS	2	Maximum number of Command Sequences that can be recursively active
EVENT_MONITOR_MAX_ASSOCIATIONS	8	Maximum number of associations that can be supported by the Event Monitor.

SYSTEM MANAGER INTERFACE

OVERVIEW

The System Manager component is responsible for the start−up and configuration of the AS0147AT device, and for managing the overall operating state of the device.

Auto−Configuration

The System Manager supports an auto−configuration feature. During system start−up, the System Manager first detects whether an NVM device is attached to the AS0147AT and if it contains a valid table of contents record. If not, the System Manager detects whether the OTPM has valid Virtual Flash record(s) or not. If no Virtual Flash records are found, the System Manager will then apply the default configuration to the sensor and hardware, and enter streaming.

The Host can disable the auto−configuration feature by grounding the SPI_SDI pin. The System Manager samples the state of this pin during the SPI device detection process. If no NVM device is detected, and the SPI_SDI pin is grounded, then auto−configuration is disabled; this is termed 'Host Configuration'.

Note: The Host can also disable or force auto−configuration by overriding the SYSMGR_−CONFIG_MODE variable default within an OTPM record. Details of the AS0147AT OTPM support are provided in "AS0147AT OTPM Contents Encoding Specification".

System State Management

The System Manager allows the Host to control the operating state of the system. The operating state determines the functionality supported by the various AS0147AT subsystems. Table 5 describes the permanent states supported by the System Manager, and Table 6 shows the functionality supported by the AS0147AT subsystems in each permanent state.

Table 5. SYSTEM MANAGER PERMANENT STATES

Name	Value	Description
SYS_STATE_IDLE	0x20	System Configuration has completed, the system is not streaming, and is waiting for commands from the Host.
SYS_STATE_STREAMING	0x31	Image data is streaming from the sensor, through the color pipe, and optionally through the graphical overlay subsystem.
SYS_STATE_SUSPENDED	0x41	All firmware operations are suspended − the hardware subsystems remain active; the sensor is in standby.
SYS_STATE_SOFT_STANDBY	0x53	The system is in soft standby (controlled through the SYSMGR_SET_STATE Host command); no subsystems are functional, with the exception of the CCI slave hardware.
SYS_STATE_HARD_STANDBY	0x5B	The system is in hard standby (controlled through the STANDBY pin); no subsystems are functional. As the part is in HARD STANDBY and the CCI buses are nonfunctional the Host can never read this state.

Table 6. SYSTEM STATE VS. SUBSYSTEM FUNCTIONALITY

System State	Idle	Streaming	Suspended	Standby
AS0147AT Subsystem				
Sensor	Standby	Active	Standby	Standby
Color Pipe	Idle	Active	Active	Standby
Sequencer	Disabled	Active	Disabled	Standby
Overlay	Disabled	Dependent upon overlay configuration	Disabled	Standby
TX subsystem	Idle	Dependent upon TX configuration	Dependent upon TX configuration	Standby
Processor subsystem	Active	Active	Active	Standby

Transient States

The System Manager also supports a number of transient states (Table 7); the system will only be in these states for a certain amount of time. The precise amount of time is dependent upon the state and system configuration.

Table 7. SYSTEM MANAGER TRANSIENT STATES

Name	Value	Description
SYS_STATE_OTPM_DEFAULTS	0x10	Processing OTPM records
SYS_STATE_OTPM_CONFIG	0x11	Locating and processing virtual Flash
SYS_STATE_OC_LOAD_DEFAULTS	0x12	Locating, verifying and applying init (defaults) table (virtual Flash)
SYS_STATE_OC_LOAD_PATCHES	0x13	Locating, verifying and loading patches (virtual Flash)
SYS_STATE_OC_LOAD_CALIB	0x14	Locating, verifying and applying calibration init table (virtual Flash)
SYS_STATE_OC_CONFIG_TASKS	0x15	Tasks/libs processing initialization tables (virtual Flash)
SYS_STATE_FLASH_CONFIG	0x17	Detecting SPI Flash/EEPROM devices
SYS_STATE_AUTO_CONFIG	0x18	Auto−configuration active
SYS_STATE_HOST_CONFIG	0x19	Host configuration active
SYS_STATE_FC_LOAD_DEFAULTS	0x1A	Locating, verifying and applying init (defaults) table (NVM)
SYS_STATE_FC_LOAD_PATCHES	0x1B	Locating, verifying and loading patches (NVM)
SYS_STATE_FC_LOAD_CALIB	0x1C	Locating, verifying and applying calibration init table (NVM)
SYS_STATE_FC_CONFIG_TASKS	0x1D	Tasks/libs processing initialization tables (NVM)
SYS_STATE_SET_FEATURES	0x21	Tasks/libs processing feature control register to determine capabilities
SYS_STATE_DO_CONFIG_CHANGE	0x29	System is actioning the Change−Config request; the current configuration is being applied to the hardware
SYS_STATE_VALIDATE_CONFIG	0x2A	System is validating that the current configuration can be applied to the hardware
SYS_STATE_HARD_STANDBY_DURING_CONFIG	0x2B	Tests STANDBY pin state and enters hard−standby if asserted. This state permits hard−standby to occur prior to entering streaming (for the first time only).
SYS_STATE_CONFIG_POWER_ON	0x2C	System is powering−on zones required for the new configuration
SYS_STATE_CONFIG_POWER_OFF	0x2D	System is powering−off zones not required for the new configuration
SYS_STATE_DO_START_STREAMING	0x30	System is starting streaming
SYS_STATE_STOP_STREAMING	0x33	System is stopping streaming
SYS_STATE_DO_SUSPEND	0x42	System is actioning the Suspend request
SYS_STATE_DO_SOFT_STANDBY	0x51	System is actioning the Soft Standby request
SYS_STATE_DRIVER_SOFT_STANDBY	0x52	Driver layer entering soft standby
SYS_STATE_DRIVER_LEAVE_SOFT_STANDBY	0x54	Driver layer leaving soft standby
SYS_STATE_LEAVE_SOFT_STANDBY	0x55	System is leaving soft standby
SYS_STATE_DO_HARD_STANDBY	0x59	System is actioning the Hard Standby request
SYS_STATE_DRIVER_HARD_STANDBY	0x5A	Driver layer entering hard standby
SYS_STATE_DRIVER_LEAVE_HARD_STANDBY	0x5C	Driver layer leaving hard standby
SYS_STATE_LEAVE_HARD_STANDBY	0x5D	System is leaving hard standby

POWER MANAGEMENT

The AS0147AT supports multiple independent power zones, to permit the firmware to power down specific elements of the hardware if they are not in use. The System Manager provides the Config Power Management host command to allow the host to configure the power state of the device.

The System Manager supports two power management modes:

1. Host−Controlled Mode
   The host can configure the desired power state of each zone. The configured power state will then take effect during the next Change−Config operation. The Change−Config operation will be

rejected with EACCES if the host has requested a zone required to be active for the configured use case should be powered off.

2. Dynamic Mode
   The firmware dynamically determines which zones should be powered on or off, based on the current configured use case. It will also power off zones during hard and soft standby (where applicable).

At start−up, all zones are powered on, and the System Manager operates in Host− Controlled mode.

SUMMARY

Table 8 summarizes the Host commands relating to the System Manager subsystem of the AS0147AT.

Table 8. SYSTEM MANAGER HOST COMMANDS

System Manager Host Command	Value	Type	Description
Set State	0x8100	Synchronous	Request the system enter a new state
Get State	0x8101	Synchronous	Get the current state of the system
Config Power Management	0x8102	Synchronous	Configure the power state of the system

SYSTEM MANAGER COMMAND PARAMETERS

Extended Type

Table 9 lists the extended parameter types specific to the System Manager subsystem.

Table 9. SYSTEM MANAGER PARAMETER TYPE

Extended Type	Size	Field	Base Type	Description
SYSMGR_STATE	8 bits	–	UINT8	State of the System Manager – see Table 10
POWER_MGMT_MODE	8 bits	–	UINT8	Power Management mode: 0: Host−controlled (default) 1: Dynamic

System Manager States

Table 10 lists the System Manager states that can be requested by the Host.

Table 10. SYSTEM MANAGER 'REQUEST' STATES

Name	Value	Description
SYS_STATE_ENTER_CONFIG_CHANGE	0x28	Requesting the 'Enter Config Change' state effects the 'Change Config' pseudo−command. It instructs the System Manager to take the system out of the streaming state, and then commands all components to reconfigure (based on the state of the firmware shared−variables). After reconfiguration, the System Manager restores the system to the streaming state.
SYS_STATE_ENTER_STREAMING	0x34	Requesting the 'Enter Streaming' state instructs the System Manager to commence streaming video through the system. The System Manager instructs all components to prepare for streaming, and then automatically enters the Streaming state.
SYS_STATE_ENTER_SUSPEND	0x40	Requesting the 'Enter Suspend' state instructs the System Manager to suspend all operations of the firmware. The System Manager instructs all components to prepare for suspension, and then automatically enters the Suspended state.
SYS_STATE_ENTER_SOFT_STANDBY	0x50	Requesting the 'Enter Soft Standby' state instructs the System Manager to place the system into soft standby. The System Manager instructs all components to prepare for soft standby, and then automatically places the system into soft standby.
SYS_STATE_LEAVE_SOFT_STANDBY	0x55	Requesting the 'Leave Soft Standby' state instructs the System Manger to exit the system from soft standby. The System Manager instructs all components to leave soft standby, and then returns the system to the state it was in before soft standby occurred.

Power Management Modes

Table 11 lists the power management modes that can be requested by the Host.

Table 11. POWER MANAGEMENT MODES

Power Management Mode	Value	Description
Host−Controlled	0x0	The host controls which power zones are powered on. This is the default mode. All power zones are powered on by default
Dynamic	0x1	The firmware automatically controls the power zones based on the configured operating use case. Zones are powered off during soft and hard standby (when appropriate).

SET STATE

This synchronous command requests the System Manager enter the specified state.

Command

Table 12.

Size	Value	Mnemonic	Description
16 bits	0x8100	SYSMGR_SET_STATE	Set the system state.

Command Parameters

Table 13.

Byte Offset	Field	Type	Value	Description
+0	STATE	SYSMGR_STATE		The desired state to enter.

RESULT STATUS

Table 14.

Result Status Code	Description
ENOERR	Command accepted and in operation.
EINVAL	Requested state parameter is invalid.
EACCES	State cannot be requested at present.
Other	Component−specific failure − see “Appendix D: Set State Command Failure Codes” on page 108.

Command Response Parameters

None.

Notes

- The Set State command requests that System Manager transition to the requested state. The time for the system to reach that state is indeterminate.
- The set of states that the Host can request at any particular time is dependent upon the current system state, as shown in Table 15. A request to enter a non−permitted state will be rejected with EACCES.

Table 15. PERMITTED STATE TRANSITIONS

Active State	Permitted State Transition	Resultant State
SYS_STATE_IDLE	SYS_STATE_ENTER_CONFIG_CHANGE	SYS_STATE_STREAMING
	SYS_STATE_ENTER_SUSPEND	SYS_STATE_SUSPENDED
	SYS_STATE_ENTER_SOFT_STANDBY	SYS_STATE_SOFT_STANDBY
SYS_STATE_STREAMING	SYS_STATE_ENTER_CONFIG_CHANGE	SYS_STATE_STREAMING
	SYS_STATE_ENTER_SUSPEND	SYS_STATE_SUSPENDED
	SYS_STATE_ENTER_SOFT_STANDBY	SYS_STATE_SOFT_STANDBY
SYS_STATE_SUSPEND	SYS_STATE_ENTER_STREAMING	SYS_STATE_STREAMING
	SYS_STATE_ENTER_CONFIG_CHANGE	SYS_STATE_STREAMING
	SYS_STATE_ENTER_SOFT_STANDBY	SYS_STATE_SOFT_STANDBY
SYS_STATE_STANDBY	SYS_STATE_LEAVE_SOFT_STANDBY	<state prior to Soft Standby>
	SYS_STATE_ENTER_CONFIG_CHANGE	SYS_STATE_STREAMING
	SYS_STATE_ENTER_SUSPEND	SYS_STATE_SUSPENDED

- The Change−Config pseudo−command − Set State (SYS_STATE_ENTER_CON− FIG_CHANGE) - is a special case, as the firmware components are permitted to reject this state change request if the firmware shared variable configuration is inappropriate. The component rejecting the request is identified through SYSMGR_CMD_− COMP_ID, and the reason code for the rejection is recorded in SYSMGR_CMD_COMP_REJECT_ID. (See “Appendix D: Set State Command Failure Codes” on page 108).

GET STATE

This synchronous command retrieves the current system state.

Table 16. COMMAND

Size	Value	Mnemonic	Description
16 bits	0x8101	SYSMGR_GET_STATE	Get the system state

Command Parameters

None.

Table 17. RESULT STATUS

Result Status Code	Description
ENOERR	Command completed successfully
EBUSY	Command processor is busy (Host Command Lock is locked)

Table 18. COMMAND RESPONSE PARAMETERS

Byte Offset	Field	Type	Value	Description
+0	STATE	SYSMGR_STATE		The current system state

Notes

The Get State command retrieves the current state of the system. Note the returned value may reflect an internal transient state.

CONFIG POWER MANAGEMENT

This synchronous command configures the power state of the system.

Table 19. COMMAND

Size	Value	Mnemonic	Description
16 bits	0x8102	SYSMGR_CONFIG_POWER_MGMT	Configure the power state of the system

Table 20. COMMAND PARAMETERS

Byte Offset	Field	Type	Value	Description
+0	Mode	POWER_MGMT_MODE	–	Desired power management mode
+1	Enable Zone A2	BOOL	FALSE: off TRUE: on	Enable power to zone A2
+2	Enable Zone A3	BOOL	FALSE: off TRUE: on	Enable power to zone A3
+3	Enable Zone A4	BOOL	FALSE: off TRUE: on	Enable power to zone A4
+4	Enable Zone A5	BOOL	FALSE: off TRUE: on	Enable power to zone A5
+5	Enable Zone A6	BOOL	FALSE: off TRUE: on	Enable power to zone A6

Table 21. RESULT STATUS

Result Status Code	Description
ENOERR	Command accepted and in operation
EINVAL	Requested parameter is invalid

Command Response Parameters

None

Notes

- This command configures the power management subsystem, but does not effect any change. The requested configuration will be applied during the next Change−Config operation.
- The ’Enable Zone An’ fields are only supported in host−controlled mode. These fields are ignored if dynamic mode is selected.
- When dynamic mode is selected, the System Manager will be configured to force a Change−Config operation to occur when leaving hard or soft standby.

OVERLAY HOST COMMAND INTERFACE

The following subsections detail the various commands to control and configure the graphical overlay function of the AS0147AT.

OVERVIEW

The overlay subsystem has four types of graphic overlays: character, bitmap image, line, and arc.

Character Overlay

Characters are 32x32 pixels. Hardware has a 192 character ROM in addition to an 8K RAM that can store 64 user characters.

Four text strings are supported. Each string holds 64 character indexes. Indexes 0 to 191 reference characters in ROM. Indexes 192 to 255 reference characters in RAM.

User character RAM data can be stored in NVM and automatically loaded after system reset.

AS0147AT has one character overlay.

Bitmap Image Overlay

The bitmap image overlay displays run−length encoded images stored in a hardware buffer.

AS0147AT has seven bitmap image overlays.

Line and Arc Overlay

The line overlay has 10 line engines and the arc overlay has 5 arc engines. Lines and arcs are drawn by configuring and enabling the engines with the Draw Shape command.

Colors for lines and arcs are stored in a 16 entry color lookup table (LUT). Each overlay has its own color LUT. Colors are in YCbCrAlpha format, 8 bits per channel. Color LUTs are loaded with the Set Color Lookup Table or Load Color Lookup Table commands.

Line Engine

AS0147AT has two line overlays to provide 20 lines and two arc overlays to provide 10 arcs.

Table 22. LINE ENGINE

Property	Description
valid	0: Line is disabled 1: Draw line
color	Index into color LUT
width	Width of line
x0 y0	Start coordinate
x1 y1	End coordinate
box	0: Draw a line 1: Draw a box with corners (x0,y0) and (x1,y1)
anti−alias	0: No anti−alias
	1: Enable anti−alias

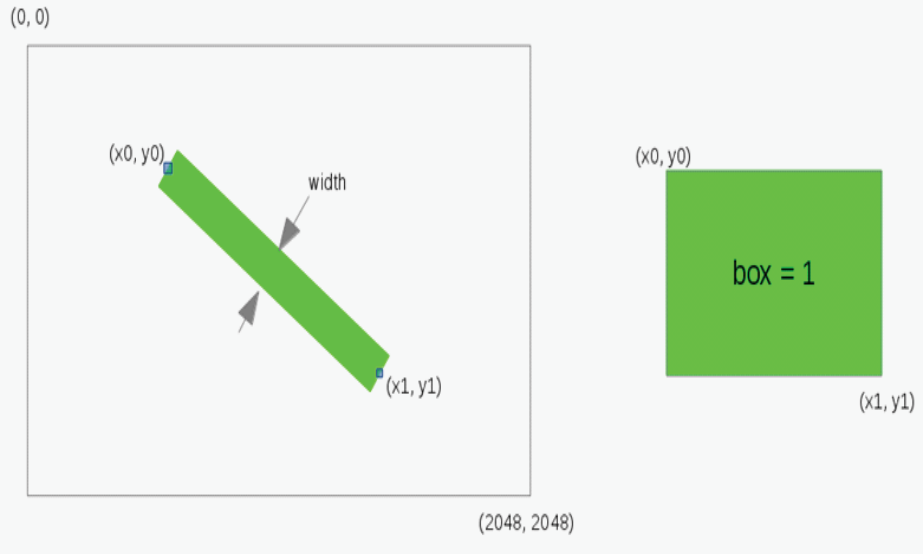

Figure 4. Line Engine

Arc Engine

Table 23. ARC ENGINE

Property	Description
valid	0: Arc is disabled 1: Draw arc
color	Index into color LUT

Table 23. ARC ENGINE

Property	Description
width	Width of arc
x y	Center coordinate
start_degrees	Angle to start arc
degrees	Length of arc
radius	Arc radius
anti−alias	0: No anti−alias 1: Draw with anti−alias

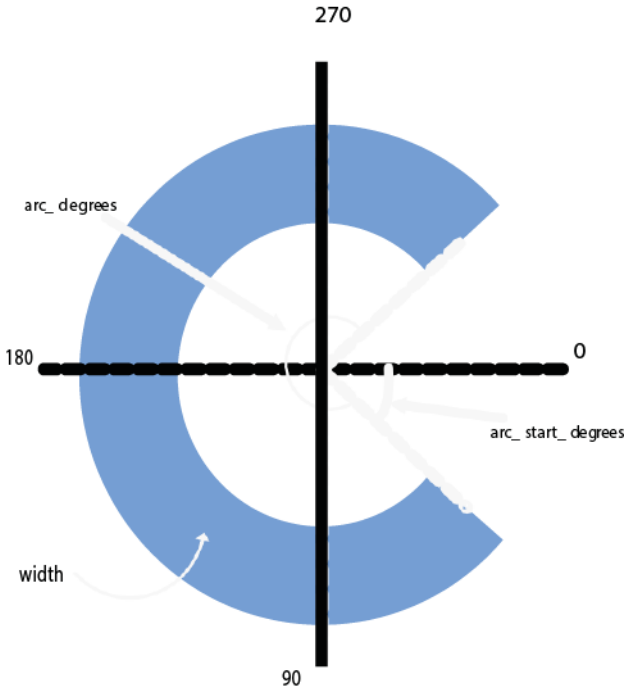

Figure 5. Arc Engine

Referencing Overlays

Overlays are numbered as shown in Table 24.

Table 24. OVERLAY NUMBERING

Overlay	Type	Overlay Name
0	bitmap	bitmap 0
1	bitmap	bitmap 1
2	bitmap	bitmap 2
3	bitmap	bitmap 3
4	bitmap	bitmap 4
5	bitmap	bitmap 5
6	bitmap	bitmap 6
7	line	line 0
8	line	line 1
9	arc	arc 0

Table 24. OVERLAY NUMBERING

Overlay	Type	Overlay Name
10	arc	arc 1
11	character	character

OVERLAY DISPLAY HIERARCHY

The hierarchy is configured by assigning overlays to different layers using the Enable Layer command. Lower number layers will be displayed on top of higher number layers.

In the following example layer configuration shown in Table 25, lines will be overlaid on top of arcs, which will be overlaid on top of bitmap images.

Table 25. EXAMPLE OVERLAY LAYER HIERARCHY

Layer	Overlay	Overlay Name
0	7	line 0
1	8	line 1
2	9	arc 0
3	10	arc 1
4	0	bitmap 0
5	1	bitmap 1
6	2	bitmap 2
7	3	bitmap 3
8	4	bitmap 4
9	5	bitmap 5
10	6	bitmap 6

Summary

Firmware always configures the character overlay to be displayed on top of the other overlays.

Table 26 summarizes the host commands relating to the graphical overlay subsystem of the AS0147AT.

Table 26. OVERLAY HOST COMMANDS

Overlay Host Command	Value	Type	Description
Enable Overlay Subsystem	0x8200	Synchronous	Enable or disable the overlay subsystem.
Get Overlay Subsystem State	0x8201	Synchronous	Retrieves the state of the overlay subsystem.
Set Calibration	0x8202	Synchronous	Set the calibration offset.
Set Bitmap Property	0x8203	Synchronous	Set a property of a bitmap.
Get Bitmap Property	0x8204	Synchronous	Get a property of a bitmap.
Set String Property0	0x8205	Synchronous	Set a property of a character string.
Set String Property1	0x8215	Synchronous	Set a property of a character string.
Set String Property2	0x8216	Synchronous	Set a property of a character string.
Set String Property3	0x8217	Synchronous	Set a property of a character string.
Load Bitmap Buffer	0x8206	Asynchronous	Load a bitmap image from NVM.
Load Status	0x8207	Synchronous	Retrieve status of an active load buffer operation.
Write Bitmap Buffer	0x8208	Synchronous	Write to a bitmap image overlay buffer.
Read Bitmap Buffer	0x8209	Synchronous	Read from a bitmap image overlay buffer.
Enable Layer	0x820A	Synchronous	Enable or disable an overlay layer.

Table 26. OVERLAY HOST COMMANDS

Overlay Host Command	Value	Type	Description
Get Layer Status	0x820B	Synchronous	Retrieve the status of an overlay layer.
Set String0	0x820C	Synchronous	Set the character string.
Set String1	0x820F	Synchronous	Set the character string.
Set String2	0x8210	Synchronous	Set the character string
Set String3	0x8211	Synchronous	Set the character string
Get String0	0x820D	Synchronous	Get the current character string
Get String1	0x8212	Synchronous	Get the current character string
Get String2	0x8213	Synchronous	Get the current character string
Get String3	0x8214	Synchronous	Get the current character string
Load String0	0x820E	Asynchronous	Load a character string from NVM
Load String1	0x821B	Asynchronous	Load a character string from NVM
Load String2	0x821C	Asynchronous	Load a character string from NVM
Load String3	0x821D	Asynchronous	Load a character string from NVM
Draw Shape	0x8218	Synchronous	Draw lines and arcs
Set Color Lookup Table	0x8219	Synchronous	Set color LUT entries for a line or arc overlay
Write User Character RAM	0x821A	Synchronous	Write to User Character RAM
Load Color Lookup Table	0x821E	Asynchronous	Load color LUT for a line or arc overlay from NVM
Load User Character RAM	0x821F	Asynchronous	Load User Character RAM from NVM

OVERLAY COMMAND PARAMETERS

Extended Types

Table 27 lists the extended parameter types specific to the graphical overlay subsystem.

Table 27. OVERLAY PARAMETER TYPES

Extended Type	Size	Field	Base Type	Description
OVRL_LUT	32 bits	–	UINT32	Overlay color look−up table [Y(8bits)−Cb(8bits)−Cr(8 bits)−Alpha(8 bits)] Additional details can be found in the AP010AT NVM Encoding Spec Technical Note TN−09−321
OVRL_CROP_MODE	8 bits	–	UINT8	Cropping mode: 0: No cropping 1: Crop outside; crop window specifies the uncropped (visible) portion of the bitmap 2: Crop inside; crop window specifies the cropped (invisible) portion of the bitmap

Overlay Properties

The graphical overlay function supports bitmap images and character strings. These share a common set of configuration properties controlled by the host. Table 19 details these properties; property identifiers are encoded within a UINT8 type.

Table 28. OVERLAY PROPERTIES

ID	Mnemonic	Type	Description
0x00	OVRL_PROP_X_POS	UINT16	Starting horizontal offset (in pixels).
0x01	OVRL_PROP_Y_POS	UINT16	Starting vertical offset (in pixels)
0x02	OVRL_PROP_X_LEN	UINT16	Width (in pixels). This property is determined by the bitmap and cannot be dynamically adjusted.
0x03	OVRL_PROP_Y_LEN	UINT16	Height (in pixels)
0x04	OVRL_PROP_SIZE	UINT16	Size of bitmap (in bytes)
0x05	OVRL_PROP_IS_CALIBRATED	FLAG	Flag to indicate if global calibration offset should be applied to the bitmap.
0x06	OVRL_PROP_LUT0	OVRL_LUT	Color LUT for color 0 / background
0x07	OVRL_PROP_LUT1	OVRL_LUT	Color LUT for color 1 / foreground
0x08	OVRL_PROP_LUT2	OVRL_LUT	Color LUT for color 2
0x09	OVRL_PROP_LUT3	OVRL_LUT	Color LUT for color 3
0x0A	OVRL_PROP_LUT4	OVRL_LUT	Color LUT for color 4
0x0B	OVRL_PROP_LUT5	OVRL_LUT	Color LUT for color 5
0x0C	OVRL_PROP_LUT6	OVRL_LUT	Color LUT for color 6
0x0D	OVRL_PROP_LUT7	OVRL_LUT	Color LUT for color 7
0x0E	OVRL_PROP_BLINK_RATE	UINT8	Bitmap blink rate (in multiples of 5 frames).
0x0F	OVRL_PROP_TIMEOUT	UINT8	Time−out (in multiples of 5 frames)
0x10	OVRL_PROP_DECIMATE	FLAG	Flag to indicate if character string font should be decimated by a factor of two
0x11	OVRL_PROP_CROP_MODE	OVRL_CROP_MODE	Bitmap cropping mode
0x12	OVRL_PROP_CROP_X_OFFSET	UINT16	Horizontal offset of crop window relative to the bitmap (in pixels)
0x13	OVRL_PROP_CROP_Y_OFFSET	UINT16	Vertical offset of crop window relative to the bitmap (in pixels)
0x14	OVRL_PROP_CROP_X_LEN	UINT16	Width of crop window (in pixels)
0x15	OVRL_PROP_CROP_Y_LEN	UINT16	Height of crop window (in pixels)
0x16	OVRL_PROP_FADER_ENABLE	FLAG	Flag to indicate if the bitmap has the fader enabled
0x17	OVRL_PROP_FADER_START	UINT16	Start fading value for the alpha. range 0 to 256: 0: Transparent 256: No change to overlay Scaled_Alpha: Alpha * current fade/256
0x18	OVRL_PROP_FADER_MAX	UINT16	Max fading value for the alpha. range 0 to 256: 0: Transparent 256: No change to overlay Scaled_Alpha: Alpha * current fade/256
0x19	OVRL_PROP_FADER_STEP	UINT16	Fading value will be incremented by fade step for each line greater than start line until fade reaches fade max. Step is a 9.4 fixed point number (4 fractional bits)
0x1A	OVRL_PROP_FADER_START_LINE	UINT16	Line number to start fading the alpha channel
0x1B	OVRL_PROP_LUT8	OVRL_LUT	Color LUT for color 8
0x1C	OVRL_PROP_LUT9	OVRL_LUT	Color LUT for color 9
0x1D	OVRL_PROP_LUT10	OVRL_LUT	Color LUT for color 10
0x1E	OVRL_PROP_LUT11	OVRL_LUT	Color LUT for color 11

Table 28. OVERLAY PROPERTIES

ID	Mnemonic	Type	Description
0x1F	OVRL_PROP_LUT12	OVRL_LUT	Color LUT for color 12
0x20	OVRL_PROP_LUT13	OVRL_LUT	Color LUT for color 13
0x21	OVRL_PROP_LUT14	OVRL_LUT	Color LUT for color 14
0x22	OVRL_PROP_LUT15	OVRL_LUT	Color LUT for color 15
0x23	OVRL_PROP_LUT16	OVRL_LUT	Color LUT for color 16
0x24	OVRL_PROP_LUT17	OVRL_LUT	Color LUT for color 17
0x25	OVRL_PROP_LUT18	OVRL_LUT	Color LUT for color 18
0x26	OVRL_PROP_LUT19	OVRL_LUT	Color LUT for color 19
0x27	OVRL_PROP_LUT20	OVRL_LUT	Color LUT for color 20
0x28	OVRL_PROP_LUT21	OVRL_LUT	Color LUT for color 21
0x29	OVRL_PROP_LUT22	OVRL_LUT	Color LUT for color 22
0x2A	OVRL_PROP_LUT23	OVRL_LUT	Color LUT for color 23
0x2B	OVRL_PROP_LUT24	OVRL_LUT	Color LUT for color 24
0x2C	OVRL_PROP_LUT25	OVRL_LUT	Color LUT for color 25
0x2D	OVRL_PROP_LUT26	OVRL_LUT	Color LUT for color 26
0x2E	OVRL_PROP_LUT27	OVRL_LUT	Color LUT for color 27
0x2F	OVRL_PROP_LUT28	OVRL_LUT	Color LUT for color 28
0x30	OVRL_PROP_LUT29	OVRL_LUT	Color LUT for color 29
0x31	OVRL_PROP_LUT30	OVRL_LUT	Color LUT for color 30
0x32	OVRL_PROP_LUT31	OVRL_LUT	Color LUT for color 31

ENABLE OVERLAY SUBSYSTEM

This synchronous command enables or disables the graphical overlay subsystem.

Command

Table 29.

Size	Value	Mnemonic	Description
16 bits	0x8200	OVRL_ENABLE	Enable (or disable) the overlay subsystem

Command Parameters

Table 30.

Byte Offset	Field	Type	Value	Description
+0	Enable	FLAG	FALSE	Disable overlay subsystem.
			TRUE	Enable overlay subsystem.

Result Status

Table 31.

Result Status Code	Description
ENOERR	Command completed successfully
EALREADY	Already enabled
EBUSY	Previous operation has not completed

Command Response Parameters

None

Notes

The firmware will synchronize the activation or deactivation of the overlay subsystem to the frame timing of the selected input stream.

GET OVERLAY SUBSYSTEM STATE

This synchronous command retrieves the status of the graphical overlay subsystem.

Command

Table 32.

Size	Value	Mnemonic	Description
16 bits	0x8201	OVRL_GET_STATE	Retrieve the state of the overlay subsystem

Command Parameters

None.

Result Status

Table 33.

Result Status Code	Description
ENOERR	Command completed successfully

Command Response Parameters

Table 34.

Byte Offset	Field	Type	Value	Description
+0	Enabled	FLAG	FALSE	Overlay disabled
			TRUE	Overlay enabled
+1	Reserved		0	Reserved
+2	Reserved		0	Reserved
+3	Error Status	UINT8		Current error status of Overlay subsystem

Notes

If the Error Status field is not ENOERR, the overlay subsystem has detected an unrecoverable error and has terminated its operations. A system restart is required.

SET CALIBRATION

This synchronous command sets the global calibration offset of the overlay subsystem.

Command

Table 35.

Size	Value	Mnemonic	Description
16 bits	0x8202	OVRL_SET_CALIBRATION	Set the calibration offset of the overlay subsystem.

Command Parameters

None.

Result Status

Table 36.

Byte Offset	Field	Type	Value	Description
+0	X_OFFSET	UINT8		Horizontal calibration offset
+1	Y_OFFSET	UINT8		Vertical calibration offset

Command Response Parameters

Table 37.

Result Status Code	Description
ENOERR	Command completed successfully
ERANGE	Either X_OFFSET or Y_OFFSET are outside the permitted calibration offset range

Notes

- The firmware will synchronize the update of the calibration offset with the underlying hardware during frame blanking to prevent image corruption.
- The calibration offset adjusts the position of all calibrated bitmaps−that is, all bitmaps with their OVRL_PROP_IS_CALIBRATED property set to TRUE. “Appendix D: Overlay Calibration” on page 108 provides more details.

SET BITMAP PROPERTY

Command

Table 38.

Size	Value	Mnemonic	Description
16 bits	0x8203	OVRL_SET_BITMAP_PROP	Set a property of a bitmap

Command Parameters

Table 39.

Byte Offset	Field	Type	Value	Description
+0	Overlay Buffer	UINT8	0...6	The buffer that contains the bitmap.
+1	Property ID	UINT8	See “Overlay Properties” on page 20	The property to modify.
+2	Pad16	UINT16		Padding for alignment
+4...	Value	–	See “Overlay Properties” on page 20	The value of the property.

Result Status

This synchronous command sets a property of a bitmap.

Table 40.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid property identifier
ENOENT	Specified buffer is empty
ERANGE	Property value is out of range
EACCES	Permission denied
EBUSY	Cannot support update at this time

Command Response Parameters

None.

Notes

- The firmware only supports property updates to one active bitmap per frame. An active bitmap is one that is being displayed by an overlay layer. The firmware will synchronize the update to the underlying hardware during frame blanking to prevent image corruption. An attempt to update properties of a second bitmap during the same frame will be rejected with EBUSY.
- An inactive bitmap property can be adjusted at any time.
- The OVRL_PROP_X_LEN bitmap property cannot be adjusted.

GET BITMAP PROPERTY

This synchronous command retrieves a property of a bitmap.

Command

Table 41.

Size	Value	Mnemonic	Description
16 bits	0x8204	OVRL_GET_BITMAP_PROP	Get a property of a bitmap

Command Parameters

Table 42.

Byte Offset	Field	Type	Value	Description
+0	Overlay Buffer	UINT8	0...6	The buffer that contains the bitmap
+1	Property ID	UINT8	See “Overlay Properties” on page 20.	The property to retrieve

Result Status

Table 43.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid property identifier
ENOENT	Specified buffer is empty

Command Response Parameters

Table 44.

Byte Offset	Field	Type	Value	Description
+0	Value	Variant	–	The value of the property

Notes

- A bitmap property can be retrieved at any time. If the bitmap is active (that is, it is being displayed by an overlay layer) AND has just been updated, then the retrieved value may represent the previous value and not the new value.
- The OVRL_PROP_TIMEOUT property represents the remaining number of frames before the bitmap times out, and NOT the initial value.

SET STRING PROPERTY

These synchronous commands set a property of an overlay character string.

Command

Table 45.

Size	Value	Mnemonic	Description
16 bits	0x8205	OVRL_SET_STRING_PROP	Set a property of character string0
16 bits	0x8215	OVRL_SET_STRING1_PROP	Set a property of character string 1
16 bits	0x8216	OVRL_SET_STRING2_PROP	Set a property of character string 2
16 bits	0x8217	OVRL_SET_STRING3_PROP	Set a property of character string 3

Command Parameters

Table 46.

Byte Offset	Field	Type	Value	Description
+0	Property ID	UINT8	See “Overlay Properties” on page 20.	The property to modify
+1	Pad8	UINT8		Padding for alignment
+2	Pad16	UIN16		Padding for alignment
+4...	Value	–	See “Overlay Properties” on page 20	The (variant) value of the property

Result Status

Table 47.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid property identifier
ERANGE	Property value is out of range

Command Response Parameters

None.

Notes

- A character string property can be adjusted at any time. The firmware will synchro− nize the update to the underlying hardware during frame blanking to prevent image corruption.
- Overlay character strings do not support the following properties:
  - OVRL_PROP_X_LEN
  - OVRL_PROP_Y_LEN
  - OVRL_PROP_SIZE
  - OVRL_PROP_IS_CALIBRATED
  - OVRL_PROP_LUT2 ... OVRL_PROP_LUT31
  - OVRL_PROP_CROP_xxx
  - OVRL_PROP_FADER_xxx

LOAD BITMAP BUFFER

This asynchronous command requests that the firmware load a bitmap image stored in NVM, and optionally display the image after loading.

Command

Table 48.

Size	Value	Mnemonic	Description
16 bits	0x8206	OVRL_LOAD_BUFFER	Load a buffer with a bitmap

Command Parameters

Table 49.

Byte Offset	Field	Type	Value	Description
+0	Overlay Buffer	UINT8	0...6	The buffer to contain the bitmap
+1	Layer	UINT8	0...10	The overlay layer on which to display the bitmap (if Enable is TRUE, otherwise ignored)
+2	Bitmap ID	UINT16		Identifies the bitmap to load
+4	Enable	FLAG		If TRUE, the bitmap will automatically be displayed when it has successfully loaded

Result Status

Table 50.

Result Status Code	Description
ENOERR	Command accepted and in operation
ENODEV	NVM device was not found – buffer not loaded
EALREADY	Attempting to enable a layer that is already active – buffer not loaded
EINVAL	Invalid overlay, layer or bitmap ID specified
EBUSY	Previous Load Bitmap Buffer or Load String request has not completed

Command Response Parameters

None.

Notes

- If the specified buffer is enabled when the command is issued, the firmware will first disable the buffer, and wait for the hardware to release it, before the new bitmap is loaded.
- The bitmap ID is determined by the OVERLAY_BITMAP_TOC record; it is the (zero−based) index of the bitmap in the OVERLAY_BITMAP_TOC.
- If a previous Load Bitmap Buffer or Load String command is still active, the request will be rejected with EBUSY.

LOAD STATUS

This synchronous command retrieves the status of a previous Load Bitmap Buffer or Load String request.

Command

Table 51.

Size	Value	Mnemonic	Description
16 bits	0x8207	OVRL_LOAD_STATUS	Get status of a Load Bitmap Buffer or Load String request

Command Parameters

None.

Result Status

Table 52.

Result Status Code	Description
ENOERR	Load Bitmap Buffer or Load String command has completed
ENOENT	Specified bitmap or string could not be located in Flash –not loaded
EIO	Bitmap or string could not be transferred –not loaded
EINVAL	Invalid buffer or string
EBUSY	Previous Load Bitmap Buffer or Load String request has not completed

Command Response Parameters

None.

Note

- The host can issue another Load Bitmap Buffer or Load String request if this command does not return EBUSY.

WRITE BITMAP BUFFER

This synchronous command requests that the firmware copies bitmap data supplied in the Parameters Pool to a bitmap buffer. This command allows a host to fill a bitmap buffer directly.

Command

Table 53.

Size	Value	Mnemonic	Description
16 bits	0x8208	OVRL_WRITE_BUFFER	Write to a bitmap buffer

Command Parameters

Table 54.

Byte Offset	Field	Type	Value	Description
+0	Overlay	UINT8	0...6	The overlay to write to
+1	Length	UINT8		Number of bytes to write (limited to PARAMS_POOL_SIZE – 4)
+2	Offset	UINT16		Offset within buffer to where data will be written
+4	Data[0]	UINT8		First byte of data
+n	Data[n−4]	UINT8		Last byte of data

Result Status

Table 55.

Result Status Code	Description
ENOERR	Command completed successfully
EALREADY	Overlay is already in use by an active layer – write not permitted
EINVAL	Invalid overlay specified
ERANGE	Offset is out−of−range
EBUSY	Previous Load Bitmap Buffer request has not completed

Command Response Parameters

None.

Notes

- It is not permitted to write to an active buffer – that is, a buffer assigned to an active layer. The host must first either disable the layer, or assign a different buffer to the layer, with the Enable Layer command.
- The firmware returns EBUSY if there is a Load Bitmap Buffer command in progress.
- The host is responsible for issuing multiple Write Bitmap Buffer requests, incrementing the Offset parameter each time.
- The bitmap data includes the RLE header fields, but excludes the configuration data. This must be written with Set Bitmap Property.

READ BITMAP BUFFER

This synchronous command requests that the firmware copies bitmap data from a bitmap overlay buffer to the Parameters Pool. This command allows a host to retrieve an overlay buffer's contents (primarily for diagnostics).

Command

Table 56.

Size	Value	Mnemonic	Description
16 bits	0x8209	OVRL_READ_BUFFER	Read from a bitmap buffer

Command Parameters

Table 57.

Byte Offset	Field	Type	Value	Description
+0	Overlay	UINT8	0...6	The overlay to read from
+1	Length	UINT8		Number of bytes to read (limited to PARAMS_POOL_SIZE)
+2	Offset	UINT16		Offset within buffer of where data will be read

Result Status

Table 58.

Result Status Code	Description
ENOERR	Command completed successfully
EALREADY	Overlay is already in use by an active layer – read not permitted
EINVAL	Invalid overlay specified
ERANGE	Offset or Length is out of range
EBUSY	Previous Load Bitmap Buffer request has not completed

Command Response Parameters

Table 59.

Byte Offset	Field	Type	Value	Description
+0	Data[0]	UINT8		First byte of data
+n	Data[n]	UINT8		Last byte of data

Notes

- It is not permitted to read from an active buffer – that is, a bitmap overlay assigned to an active layer. The host must first either disable the layer, or assign a different overlay to the layer, with the Enable Layer command.
- The firmware cannot support a Read Bitmap Buffer operation while an active buffer load (whether from NVM or directly by the host) is in progress.
- The bitmap data includes the RLE header fields, but excludes the configuration data. This must be read with Get Bitmap Property

ENABLE LAYER

This synchronous command instructs the firmware to enable or disable an overlay layer.

Command

Table 60.

Size	Value	Mnemonic	Description
16 bits	0x820A	OVRL_ENABLE_LAYER	Enable (or disable) an overlay layer

Command Parameters

Table 61.

Byte Offset	Field	Type	Value	Description
+0	Layer	UINT8	0...10	The layer to enable or disable
+1	Buffer	UINT8	0...10	The overlay to be displayed
+2	Enable	FLAG		If TRUE, the overlay will be displayed on Layer (in Byte offset 0)

Result Status

Table 62.

Result Status Code	Description
ENOERR	Command completed successfully
ENOENT	Specified bitmap image overlay is empty
EALREADY	Layer is already disabled (if Enable is FALSE)
EINVAL	Invalid overlay is specified
EBUSY	Device busy with another operation

Command Response Parameters

None.

Notes

- A layer can be enabled or disabled at any time. The firmware will synchronize the update to the underlying hardware during frame blanking to prevent image corruption.
- The overlay being displayed on each layer can be changed without requiring the layer to be disabled – the Enable Layer command can simply be reissued with a different buffer number specified.
- If a Layer is assigned an overlay that is currently used by another layer, this layer will be automatically disabled. Two layers cannot use the same overlay.
- Line and arc overlays (7, 8, 9, and 10) should be assigned only to layers 0 to 6.

GET LAYER STATUS

This synchronous command retrieves the status of an overlay layer.

Command

Table 63.

Size	Value	Mnemonic	Description
16 bits	0x820B	OVRL_GET_LAYER_STATUS	Retrieve the status of a layer.

Command Parameters

Table 64.

Byte Offset	Field	Type	Value	Description
+0	Layer	UINT8	0...10	The layer to query

Result Status

Table 65.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid layer specified

Command Response Parameters

Table 66.

Byte Offset	Field	Type	Value	Description
+0	Enable	FLAG		Indicates if layer is enabled.
+1	Overlay	UINT8	0...10	If 'Enable' is TRUE, indicates the overlay being displayed
+2	Error	UINT8		Hardware−specific error status

Notes

The Error field of the Command Response encodes a hardware−specific error status that may help diagnose display problems.

SET STRING

These synchronous commands set the overlay character string.

Command

Table 67.

Size	Value	Mnemonic	Description
16 bits	0x820C	OVRL_SET_STRING0	Set overlay character string0
16 bits	0x820F	OVRL_SET_STRING1	Set overlay character string 1
16 bits	0x8210	OVRL_SET_STRING2	Set overlay character string 2
16 bits	0x8211	OVRL_SET_STRING3	Set overlay character string 3

Command Parameters

Table 68.

Byte Offset	Field	Type	Value	Description
+0	Enable15_0	BITFIELD_16		Enable mask for character positions 15 to 0
+2	Enable31_16	BITFIELD_16		Enable mask for character positions 31 to 16
+4	Enable47_32	BITFIELD_16		Enable mask for character positions 47 to 32
+6	Enable63_48	BITFIELD_16		Enable mask for character positions 63 to 48
+8	CharIndex_0	UINT8	0 – 255	Index of character at position 0
+9	CharIndex_1	UINT8	0 – 255	Index of character at position 1
...				
+71	CharIndex_63	UINT8	0 – 255	Index of character at position 63

Result Status

Table 69.

Result Status Code	Description
EBUSY	Previous Load Bitmap Buffer or Load String command has not completed.
ENOERR	Command completed successfully

Command Response Parameters

None.

Notes

- The overlay character string can be updated at any time. The firmware will synchronize the update to the underlying hardware during frame blanking to prevent image corruption.
- The character string can contain up to 64 characters, each enabled separately.
- Character indexes 0 to 191 are detailed in “Appendix E: Character String Font” on page 110.
- Character indexes 192 to 255 are in User Character RAM.
- It is not possible to perform a partial update to the character string.

GET STRING

These synchronous commands get the current overlay character string.

Command

Table 70.

Size	Value	Mnemonic	Description
16 bits	0x820D	OVRL_GET_STRING0	Get overlay character string 0
16 bits	0x8212	OVRL_GET_STRING1	Get overlay character string 1
16 bits	0x8213	OVRL_GET_STRING2	Get overlay character string 2
16 bits	0x8214	OVRL_GET_STRING3	Get overlay character string 3

Table 71.

Result Status Code	Description
ENOERR	Command completed successfully
EBUSY	Previous Load Bitmap Buffer or Load String request has not completed

Command Parameters Result Status

These synchronous commands get the current overlay character string.

Command Response Parameters

None

Table 72.

Byte Offset	Field	Type	Value	Description
+0	Enable15_0	BITFIELD_16		Enable mask for character positions 15 to 0
+2	Enable31_16	BITFIELD_16		Enable mask for character positions 31 to 16
+4	Enable47_32	BITFIELD_16		Enable mask for character positions 47 to 32
+6	Enable63_48	BITFIELD_16		Enable mask for character positions 63 to 48
+8	CharIndex_0	UINT8	0 – 255	Index of character at position 0
+9	CharIndex_1	UINT8	0 – 255	Index of character at position 1
...				
+71	CharIndex_63	UINT8	0 – 255	Index of character at position 63

Notes

The available characters are detailed in Appendix D: Set State Command Failure Codes.

LOAD STRING

These asynchronous commands request that the firmware loads and displays an overlay string stored in NVM.

Command

Table 73.

Size	Value	Mnemonic	Description
16 bits	0x820E	OVRL_LOAD_STRING0	Load string0
16 bits	0x821B	OVRL_LOAD_STRING1	Load string 1
16 bits	0x821C	OVRL_LOAD_STRING2	Load string 2
16 bits	0x821D	OVRL_LOAD_STRING3	Load string 3

Command Parameters

Table 74.

Byte Offset	Field	Type	Value	Description
+0	String ID	UINT16		Identifies the string to load

Result Status

Table 75.

Result Status Code	Description
ENOERR	Command accepted and in operation
ENODEV	NVM device was not found – string not loaded
EINVAL	Invalid string ID specified
EBUSY	Previous Load Bitmap Buffer or Load String request has not completed

Command Response Parameters

None.

Notes

- The string ID is determined by the OVERLAY_STRING_TOC record; it is the (zero–based) index of the string in the OVERLAY_STRING_TOC.
- If a previous Load Bitmap Buffer or Load String command is still active, the request will be rejected with EBUSY.

DRAW SHAPE

This synchronous command configures line and arc overlay engines to draw lines and arcs.

Command

Table 76.

Size	Value	Mnemonic	Description
16 bits	0x8218	OVRL_DRAW_SHAPE	Draw lines and arcs

Command Parameters

The parameters for this command are descriptors to describe the lines and arcs to be drawn.

Multiple lines and arcs can be drawn by concatenating the descriptors together to form a single command parameter set. The byte immediately following the last descriptor must be 0 to terminate the parameter set.

Table 77.

Parameter Set
line/arc descriptor 0
line/arc descriptor 1
...
line/arc descriptor n
0x00

The entire length of the parameter set must not exceed PARAMS_POOL_SIZE. Specifications for the descriptors are in the Notes for this section.

Result Status

Table 78.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid line or arc descriptor

Command Response Parameters

None

Notes

- Firmware will calculate hardware registers to generate lines and arcs, then update them during frame blanking.
- Calibration offsets do not affect line and arc overlays.
- Coordinate system for overlays use (0, 0) as top left corner.

Line Descriptor Details

The following properties are required to create a line descriptor for Line Engine:

- uint8 line_overlay // which line overlay to use, legal values 0 or 1
- uint8 engine // which line engine to use, legal values 0 to 9
- uint8 width // line width, legal values 1 to 16
- uint8 color // use YCbCrAlpha value at color_LUT[color]
- uint16 x0, y0 // start pixel coordinate
- uint16 x1, y1 // end pixel coordinate
- bool valid // if true, line is drawn, otherwise line is disabled
- bool box// if true, a box is drawn with corners (x0,y0) and (x1,y1)
- bool anti_alias // if true, draw with anti−aliasing enabled
- uint16 control = (valid << 9) | (box << 10) | (anti_alias << 8) | ((width−1) << 4) | color

Table 79.

Byte	Value	Description
0	0xC1	line descriptor type
1	(line_overlay << 4) | engine	line_overlay = 0 or 1, engine = 0...9
2	control >> 8	upper byte of control
3	control & 0xFF	lower byte of control
4	x0 >> 8	upper byte of x0
5	x0 & 0xFF	lower byte of x0
6	y0 >> 8	upper byte of y0
7	y0 & 0xFF	lower byte of y0
8	x1 >> 8	upper byte of x1
9	x1 & 0xFF	lower byte of x1
10	y1 >> 8	upper byte of y1
11	y1 & 0xFF	lower byte of y1

Arc Descriptor Details

The following properties are required to create an arc descriptor for Arc Engine:

- uint8 arc_overlay // which arc overlay to use, legal values 0 or
- uint8 engine // which arc engine to use, legal values 0 to 4
- uint8 width // arc width, legal values 1 to 16
- uint8 color // use YCbCrAlpha value at color_LUT[color]
- int32 x, y // arc center pixel coordinate
- float start_degrees // start angle in degrees
- float degrees // length of arc, positive = clockwise, negative = counter−clockwise
- uint16 radius // legal values 1 to 65535
- bool valid // if true, arc is drawn, otherwise arc is disabled
- bool anti_alias // if true, draw with anti−aliasing enabled
- uint16 control = (valid << 9) | (anti_alias << 8) | ((width−1) << 4) | color

Table 80.

Byte	Value	Description
0	0xC2	arc descriptor type
1	(arc_overlay << 4) | engine	arc_overlay = 0 or 1, engine = 0...4
2	control >> 8	upper byte of control
3	control & 0xFF	lower byte of control
4	x >> 24	bits 31:24 of x
5	x >> 16	bits 23:16 of x
6	x >> 8	bits 15:8 of x
7	x & 0xFF	bits 7:0 of x
8	y >> 24	bits 31:24 of y
9	y >> 16	bits 23:16 of y
10	y >> 8	bits 15:8 of y
11	y & 0xFF	bits 7:0 of y
12	radius >> 8	upper byte of radius
13	radius & 0xFF	lower byte of radius
14	start_degrees >> 24	bits 31:24 of start_degrees

Table 80.

Byte	Value	Description
15	start_degrees >> 16	bits 23:16 of start_degrees
16	start_degrees >> 8	bits 15:8 of start_degrees
17	start_degrees & 0xFF	bits 7:0 of start_degrees
18	degrees >> 24	bits 31:24 of degrees
19	degrees >> 16	bits 23:16 of degrees
20	degrees >> 8	bits 15:8 of degrees
21	degrees & 0xFF	bits 7:0 of degrees

SET COLOR LOOKUP TABLE

This synchronous command sets the colors in a color LUT for line and arc overlays. Line and arc overlays have a color LUT with 16 entries each.

Command

Table 81.

Size	Value	Mnemonic	Description
16 bits	0x8219	OVRL_SET_COLOR_LUT	Set color LUT entries

Command Parameters

Table 82.

Byte Offset	Field	Type	Value	Description
+0	Type	UINT8	0 or 1	0: Line 1: Arc
+1	Number	UINT8	0 or 1	Line or arc overlay number
+2	Start_Index	UINT8	0 to 15	Set colors starting at LUT Start_Index]
+3	Entries	UINT8	1 to 16	Number of colors to set
+4	Color 0	UINT32		Y, Cb, Cr, Alpha
+8	Color 1	UINT32		Y, Cb, Cr, Alpha
...	Color (Entries−1)	UINT32		Y, Cb, Cr, Alpha

Result Status

Table 83.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid Start_Index or Entries

Command Response Parameters

None

Notes

- To set color LUT for line 0 overlay: Type = 0, Number = 0
- To set color LUT for line 1 overlay: Type = 0, Number = 1
- To set color LUT for arc 0 overlay: Type = 1, Number = 0
- To set color LUT for arc 1 overlay: Type = 1, Number = 1

WRITE USER CHARACTER RAM

This synchronous command writes data to User Character RAM.

Command

Table 84.

Size	Value	Mnemonic	Description
16 bits	0x821A	OVRL_WRITE_USER_CHAR	Write to User Character RAM

Command Parameters

Table 85.

Byte Offset	Field	Type	Value	Description
+0	Offset	UINT16		Offset in User Character RAM to where data will be written
+2	Length	UINT16		Number of bytes to write (limited to PARAMS_POOL_SIZE – 4)
+4	Data[0]	UINT8		First byte of data
+5	Data[1]	UINT8		Second byte of data
...				

Result Status

Table 86.

Result Status Code	Description
ENOERR	Command completed successfully
EBUSY	Overlay subsystem is enabled
EINVAL	Invalid Offset or Length

Command Response Parameters

None

Notes

The overlay subsystem must be disabled before issuing this command.

User character RAM can store 64 32x32 pixel characters. Each character uses 128 bytes. The offset for the Nth user character is N * 128, where N=0 to 63.

User characters are referenced by using indexes 192 – 255 in the Set String command.

- index 192 = user character at offset 0
- index 193 = user character at offset 128

The 32x32 character uses 1 bit per−pixel.

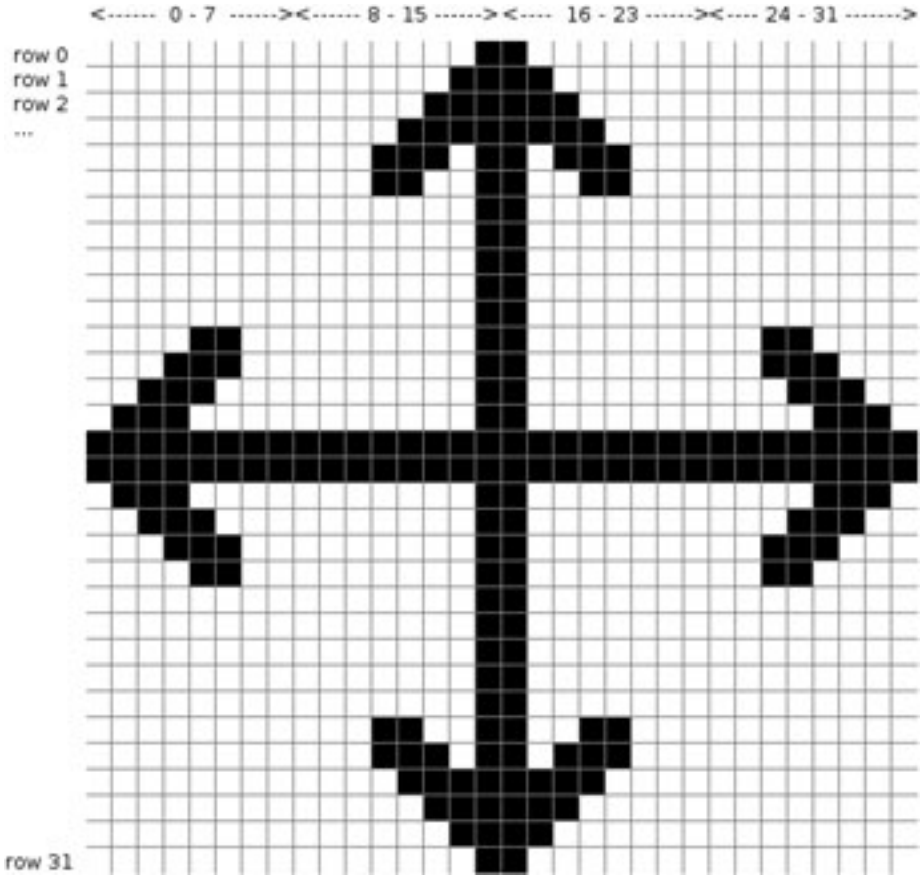

Figure 6. User Character RAM Example

Data for user character shown above is uint8 user_char[128]= {

// columns

0–7	8–15	16–23	24–31	
0x00,	0x01,	0x80,	0x00	//row 0
0x00,	0x03,	0xC0,	0x00	//row 1
0x00,	0x07,	0xE0	, 0x00	//row 2
0x00,	0x0F,	0xF0	,0x00	’/row 3
0x00,	0x1D,	0xB8,	0x00	’/row 4
0x00,	0x19,	0x98,	0x00	r/row 5
0x00,	0x01,	0x80,	0x00	//row 6
0x00,	0x01,	0x80,	0x00	//row 7
0x00,	0x01,	0x80,	0x00	// row 8
0x00,	0x01,	0x80,	0x00	// row 9
0x00,	0x01,	0x80,	0x00	// row 10
0x0C,	0x01,	0x80,	0x30	// row 11
0x1C,	0x01,	0x80,	0x38	// row 12
0x38,	0x01,	0x80,	0x1C	// row 13

```
0x70,   0x01,   0x80,   0x0E   // row 14
0xFF,   0xFF,   0xFF,   0xFF   // row 15
0xFF,   0xFF,   0xFF,   0xFF   // row 16
0x70,   0x01,   0x80,   0x0E   // row 17
0x38,   0x01,   0x80,   0x1C   // row 18
0x1C,   0x01,   0x80,   0x38   // row 19
0x0C,   0x01,   0x80,   0x30   // row 20
0x00,   0x01,   0x80,   0x00   // row 21
0x00,   0x01,   0x80,   0x00   // row 22
0x00,   0x01,   0x80,   0x00   // row 23
0x00,   0x01,   0x80,   0x00   // row 24
0x00,   0x01,   0x80,   0x00   // row 25
0x00,   0x19,   0x98,   0x00   // row 26
0x00,   0x1D,   0xB8,   0x00   // row 27
0x00,   0x0F,   0xF0,   0x00   // row 28
0x00,   0x07,   0xE0,   0x00   // row 29
0x00,   0x03,   0xC0,   0x00   // row 30
0x00,   0x01,   0x80,   0x00   // row 31
};
```

LOAD COLOR LOOKUP TABLE

This asynchronous command loads a color LUT stored in NVM into a line or arc overlay.

Command

Table 87.

Size	Value	Mnemonic	Description
16 bits	0x821E	OVRL_LOAD_COLORLUT	Load color LUT entries

Command Parameters

Table 88.

Byte Offset	Field	Type	Value	Description
+0	Type	UINT8	0 or 1	0: Line 1: Arc
+1	Number	UINT8	0 or 1	Line or arc overlay number
+2	Table ID	UINT16		Identifies the color LUT to load

Result Status

Table 89.

Result Status Code	Description
ENOERR	Command accepted and in operation
ENODEV	NVM device was not found – table not loaded
EINVAL	Invalid Table ID specified
EBUSY	A previous Load request has not completed

Command Response Parameters

None

Notes

- The table ID is determined by the OVERLAY_COLORLUT_TOC record - it is the (zero–based) index of the color LUT in the OVERLAY_COLORLUT_TOC.
- To load color LUT for line 0 overlay: Type = 0, Number = 0
- To load color LUT for line 1 overlay: Type = 0, Number = 1
- To load color LUT for arc 0 overlay: Type = 1, Number = 0
- To load color LUT for arc 1 overlay: Type = 1, Number = 1

LOAD USER CHARACTER RAM

This asynchronous command loads data into User Character RAM from NVM.

Command

Table 90.

Size	Value	Mnemonic	Description
16 bits	0x821F	OVRL_LOAD_USER_CHAR	Load User Character RAM

Command Parameters

Table 91.

Byte Offset	Field	Type	Value	Description
+0	Character RAM ID	UINT16		Identifies the RAM image to load

Result Status

Table 92.

Result Status Code	Description
ENOERR	Command accepted and in operation
ENODEV	NVM device was not found – table not loaded
EINVAL	Invalid Character RAM ID specified
EBUSY	A previous Load request has not completed or overlay is enabled

Command Response Parameters

None

Notes

The Character RAM ID is determined by the OVERLAY_USERCHAR_TOC record - it is the (zero−based) index of the character data in the OVERLAY_USERCHAR_TOC.

SPATIAL TRANSFORM ENGINE INTERFACE

The following subsections detail the various commands to control and configure the Spatial Transform Engine (STE) function of the AS0147AT.

OVERVIEW

The STE Manager firmware component controls the operation of the STE function. The firmware uses a configuration blob to configure the system for STE operation. Configu− ration blobs are generated by the STE GUI tool provided by ON Semiconductor. Each blob contains the necessary data to configure the sensor, PLLs, STE hardware and other components of the system.

The host can request a 'configuration blob' to be loaded from a number of sources:

- From non−volatile memory (OTPM or NVM) under firmware control.
- From a set of defaults stored in ROM, under firmware control.
- Through the Write Config host command (transferred by the host through the CCI).

On loading, configuration blobs are stored within the firmware's configuration cache. Note that the configuration is not applied to the system automatically on successful completion of the operation. The configuration is applied during the next Change−Config operation subsequent to the load.

SUMMARY

Table 93 summarizes the host commands relating to the STE subsystem of the AS0147AT.

Table 93. STE MANAGER HOST COMMANDS

STE Manager Host Command	Value	Type	Description
Config	0x8310	Synchronous	Configure using the default NTSC or PAL configuration stored in ROM
Load Config	0x8311	Asynchronous	Load a configuration from NVM to the configuration cache.
Load Status	0x8312	Synchronous	Get status of a Load Config request
Write Config	0x8313	Synchronous	Write configuration (through the CCI) to the configuration cache.

Config

This synchronous command requests that the firmware configure STE using the default configuration stored in ROM.

Command

Table 94.

Result Status Code	Description
ENOERR	Command completed successfully
ENOENT	No configuration in ROM for this sensor, or sensor unknown.
EBUSY	Previous Load Config request has not completed

Command Parameters

None

Result Status

Table 95.

Size	Value	Mnemonic	Description
16 bits	0x8310	STE_CONFIG	Configure using the default configuration stored in ROM

Command Response Parameters

None

Notes

- The firmware uses the CAM_SENSOR_CONTROL_MODEL_ID firmware variable to determine which configuration to load. The command will fail with ENOENT if the sensor is unknown (sensor discovery has not been performed), or the sensor model or revision is not supported.

LOAD CONFIG

This asynchronous command requests that the firmware initiate a configuration load from NVM. A subsequent Change−Config operation is required to complete the operation.

Command

Table 96.

Size	Value	Mnemonic	Description
16 bits	0x8311	STE_LOAD_CONFIG	Load a configuration from NVM to the configuration cache

Command Parameters

Table 97.

Byte Offset	Field	Type	Value	Description
+0	Config ID	UINT16		Identifies the configuration set

Result Status

Table 98.

Result Status Code	Description
ENOERR	Command accepted and in operation
EINVAL	Configuration set specified by Config ID is not valid
ENODEV	NVM device was not found − configuration not loaded
EBUSY	Previous load operation still active.

Command Response Parameters

None

Notes

- The configuration set IDs are determined by the STE_CONFIG_TOC record - the Config ID is the index of the desired configuration set in the STE_CONFIG_TOC.
- If a previous STE_LOAD_CONFIG operation is still active, the request will be rejected with EBUSY.
- If the configuration set specified by the Config ID parameter is invalid (not within the range of the STE_CONFIG_TOC) the request will be rejected with EINVAL.
- The host should use the Load Status command to determine when the request completes, and the completion status.

LOAD STATUS

This synchronous command retrieves the status of a previous Load Config request.

Command

Table 99.

Result Status Code	Description
ENOERR	Load Config command has completed
ENOENT	Specified configuration sets could not be located in NVM − configuration not loaded
ENOMEM	Cannot allocate enough internal memory to contain configuration set
EIO	Configuration set could not be transferred − configuration not loaded
EBUSY	Previous Load Config request has not completed

Command Parameters Result Status

None

Table 100.

Size	Value	Mnemonic	Description
16 bits	0x8312	STE_LOAD_STATUS	Get status of a Load Config request

Command Response Parameters

None

Notes

- The host can issue another Load Config request only if this command does not return EBUSY.

WRITE CONFIG

This synchronous command requests that the firmware copies configuration data supplied in the Parameters Pool to the internal configuration cache.

Command

Table 101.

Size	Value	Mnemonic	Description
16 bits	0x8313	STE_WRITE_CONFIG	Write a configuration (via CCI) to the configuration cache

Command Parameters

Table 102.

Byte Offset	Field	Type	Value	Description
+0	Data[0]	UINT8		First byte of data
+n	Data[n−1]	UINT8		Last byte of data

Result Status

Table 103.

Result Status Code	Description
ENOERR	Command completed successfully
ENOMEM	Cannot allocate enough internal memory to support requested transfer. (The first write command contains total length of data to be supplied)
EBUSY	Previous Load Config request has not completed

Command Response Parameters

None

Notes

- This command allows a host to write to the configuration cache directly. The data within the first write command contains the total length of data to be supplied (encoded within the data stream).
- Data must be supplied by the host in blocks of PARAMS_POOL_SIZE bytes in a sequential manner. The last write can be shorter than PARAMS_POOL_SIZE but contain all the bytes in the configuration set.
- If the first Write Config command is successful (completes with ENOERR) then the host MUST complete the full sequence of writes. During the write operation the STE subsystem is locked out from performing any other load or re−configuration tasks. There is no way to abort a write once it has commenced.
- The configuration data encoding is identical to the payload of the STE_CONFIG_V2 record.

GPIO HOST INTERFACE

OVERVIEW

The firmware supports up to 32 logical general−purpose I/O pins. The mapping between logical pin and physical pin is product−specific. Each logical pin can be assigned an owner, and a direction. The firmware will determine whether the underlying hardware can support the desired logical pin configuration.

Output Pin Ownership

At any particular time, the state of an input, output, or input/output pin is exclusively controlled by either:

- The AS0147AT hardware
  The pin is controlled by a hardware function (or firmware) within the AS0147AT, for example, the trigger pin on GPIO0.
- The Host
  The pin is controlled by the Host.

Note: The Host can sample an input pin state at any time, regardless of its ownership.

GPIO Pin Mapping

Table 104 shows the mapping between the AS0147AT physical pins and the logical GPIO pins as supported by the GPIO Manager.

Table 104. GPIO PIN MAPPING

Pin (Name)	Type	Mapping	Alternate Function
GPIO_0	Input/Output	GPIO0	Trigger Out (output)
GPIO_1	Input/Output	GPIO1	–
GPIO_2	Input/Output	GPIO2	–
GPIO_3	Input/Output	GPIO3	–
GPIO_4	Input/Output	GPIO4	–
GPIO_5	Input/Output	GPIO5	–
GPIO_6	Input/Output	GPIO6	–
Unassigned	–	GPIO7..GPIO31	–

General Purpose Input Monitoring

The AS0147AT firmware implements a general purpose (GP) input monitoring function. Host Command Sequences can be associated with a pin mask/value pair. When the masked pin state matches that of a registered value, the firmware will load and process the associated COMMAND_SEQ_Vx record (stored in NVM).

The firmware samples the GPIO pin state every GPIO_GPI_SAMPLE_PERIOD milliseconds, and evaluates each enabled association (the firmware supports GPIO_MAX_G− PI_ASSOCIATIONS associations). Note that the firmware implements a 'de−bounce' feature, so an association will only become active if the pin state is identical for two successive samples.

The firmware continues GP input sampling while the COMMAND_SEQ_Vx record is being processed (to implement the de−bounce feature). However, no further associations are evaluated during this time. Therefore, a subsequent COMMAND_SEQ_Vx record will not be processed until the next sample period following completion of a prior sequence.

SUMMARY

Table 105 summarizes the Host commands relating to the GPIO subsystem of the AS0147AT.

Table 105. GPIO HOST COMMANDS

GPIO Host Command	Value	Type	Description
Set GPIO Property	0x8400	Synchronous	Set a property of one or more GPIO pins
Get GPIO Property	0x8401	Synchronous	Retrieve a property of a GPIO pin
Set GPIO State	0x8402	Synchronous	Set the state of a GPO pin or pins
Get GPIO State	0x8403	Synchronous	Get the state of a GPI pin or pins
Set GPIO Association	0x8404	Synchronous	Associate a GPIO pin state with a Command Sequence stored in NVM
Get GPIO Association	0x8405	Synchronous	Retrieve a GPIO pin association

GPIO COMMAND PARAMETERS

Extended Types

Table 106 lists the extended parameter types specific to the GPIO subsystem.

Table 106. GPIO PARAMETER TYPES

Extended Type	Size	Field	Base Type	Description
GPIO_OWNER	8 bits	–	UINT8	Owner of the GPIO pin: 0: Not owned by Host 1: Owned by Host
GPIO_DIRECTION	8 bits	–	UINT8	Direction of the GPIO pin: 0: Output 1: Input
GPIO_ENABLE	8 bits	–	UINT8	Enable a GPIO pin group: 0: Disabled 1: Enabled

GPIO Properties

The GPIO Manager supports properties of each GP input and/or output. Table 107 details these properties; property identifiers are encoded within a UINT8 type.

Table 107. GPIO PROPERTIES

ID	Mnemonic	Type	Description
0x00	GPIO_PROP_OWNER	GPIO_OWNER	Owner of the pin
0x01	GPIO_PROP_DIRECTION	GPIO_DIRECTION	Direction of the pin
0x02	GPIO_PROP_ENABLE	GPIO_ENABLE	Enable or disable a pin

SET GPIO PROPERTY

This synchronous command sets a property of GPIO pins.

Command

Table 108.

Size	Value	Mnemonic	Description
16 bits	0x8400	GPIO_SET_PROP	Set property of GPIO pins.

Command Parameters

Table 109.

Byte Offset	Field	Type	Value	Description
+0	Mask	BITFIELD_32		The mask of logical pins to set (set bits in mask map to logical pins to configure)
+4	Property ID	UINT8	See “GPIO Properties” on page 49	The property to modify
+5	Value	UINT8	See “GPIO Properties” on page 49	The value of the property

Result Status

Table 110.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid property identifier
ENOSYS	Mask specifies unsupported pins
EACCES	Host does not own the pin(s), or pins are already in use internally.
ERANGE	Property value is out of range

Command Response Parameters

None.

Notes

- A GPIO property can be adjusted at any time.

GET GPIO PROPERTY

This synchronous command retrieves a property of a GPIO pin.

Command

Table 111.

Size	Value	Mnemonic	Description
16 bits	0x8401	GPIO_GET_PROP	Get a property of a pin

Command Parameters

Table 112.

Byte Offset	Field	Type	Value	Description
+0	Mask	BITFIELD_32		The mask of the logical pin to interrogate
+4	Property ID	UINT8	See “GPIO Properties” on page 49	The property to retrieve

Result Status

Table 113.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid property identifier
ERANGE	Mask contains more than one pin identifier

Command Response Parameters

Table 114.

Byte Offset	Field	Type	Value	Description
+0	Value	Variant	–	The value of the property

Notes

None.

SET GPIO STATE

This synchronous command sets the state of GPIO pins that are configured for output.

Command

Table 115.

Size	Value	Mnemonic	Description
16 bits	0x8402	GPIO_SET_STATE	Set the state of GPIO pins.

Command Parameters

Table 116.

Byte Offset	Field	Type	Value	Description
+0	Mask	BITFIELD_32		The mask of logical pins to set (set bits in mask map to logical pins to set)
+4	State	BITFIELD_32		The state of the logical pins (after Mask is applied)

Result Status

Table 117.

Result Status Code	Description
ENOERR	Command completed successfully
EACCES	Host does not own specified output pin(s)
ENOSYS	Mask specifies unsupported pin(s)

Command Response Parameters

None.

Notes

- A Host−owned GPIO pin (configured for output) can be adjusted at any time.

GET GPIO STATE

This synchronous command retrieves the state of GPIO pins (configured for output or input).

Command

Table 118.

Size	Value	Mnemonic	Description
16 bits	0x8403	GPIO_GET_STATE	Get the state of GPIO pins.

Command Parameters

Table 119.

Byte Offset	Field	Type	Value	Description
+0	Mask	BITFIELD_32		The mask of the logical pins to interrogate

Result Status

Table 120.

Result Status Code	Description
ENOERR	Command completed successfully
ENOSYS	Mask specifies unsupported pin(s)

Command Response Parameters

Table 121.

Byte Offset	Field	Type	Value	Description
+0	Value	BITFIELD_32		The GPIO pin state (after Mask is applied)

Notes

- The Host can query the state of any GPIO pin (configured for input or output) at any time.

SET GPIO ASSOCIATION

This synchronous command instructs the firmware to enable or disable a GPIO pin association with a Command Sequence stored in NVM.

Command

Table 122.

Size	Value	Mnemonic	Description
16 bits	0x8404	GPIO_SET_GPI_ASSOC	Set GPIO association with a Command Sequence.

Command Parameters

Table 123.

Byte Offset	Field	Type	Value	Description
+0	Association ID	UINT8	0 ... m−1	Association to modify (where m is GPIO_MAX_GPI_ASSOCIATIONS)
+1	Enable	FLAG		TRUE to enable the association, FALSE to disable
+2	Sequence ID	UINT16		Identifies the Command Sequence to load and process
+4	GPIO mask	UINT32		Identifies the GPIO bits that define this association
+8	GPIO state	UINT32		The state of the GPIO bits that will activate this association

Result Status

Table 124.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid association ID
ENOENT	No association exists (if Enable is FALSE)
ENODEV	Specified GPIO mask cannot be supported
EBUSY	A recently disabled association is still executing and can not be modified yet. Try again, or try a different association id.
EACCES	Host does not own appropriate pin group
EALREADY	Association is already in use

Command Response Parameters

None.

Notes

- The firmware will support GPIO_MAX_GPI_ASSOCIATIONS associations between GPIO states and a Command Sequence.
- The Sequence ID command parameter is not validated by the firmware.
- The GPIO Mask and GPIO State command parameters define which GPIO pins will be monitored by the firmware, and the state of those pins that will activate the association. Any other state will deactivate the association.
- When an association is deactivated, the firmware will take no action.
- When an association is activated, the firmware will instruct the Command Handler task to load and process the specified COMMAND_SEQ_Vx record from NVM.
- If a GPIO state change occurs such that multiple associations become active, the firm− ware will process these associations in sequential order, beginning from association 0.

GET GPIO ASSOCIATION

This synchronous command retrieves the state of a GPIO association.

Command

Table 125.

Size	Value	Mnemonic	Description
16 bits	0x8405	GPIO_GET_GPI_ASSOC	Get GPIO association state

Command Parameters

Table 126.

Byte Offset	Field	Type	Value	Description
+0	Association ID	UINT8	0 .. m−1	Association to retrieve (where m is GPIO_MAX_GPI_ASSOCIATIONS)

Result Status

Table 127.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid association ID

Command Response Parameters

Table 128.

Byte Offset	Field	Type	Value	Description
+0	Association ID	UINT8	0 .. m−1	Association retrieved (where m is GPIO_MAX_GPI_ASSOCIATIONS)
+1	Enable	FLAG		TRUE if association is enabled, FALSE if disabled
+2	Sequence ID	UINT16		Identifies the Command Sequence to load and process (if Enable is TRUE)
+4	GPIO Mask	UINT32		Identifies the GPIO bit that defines this association (if Enable is TRUE)
+8	GPIO State	UINT32		The state of the GPIO bit that will activate this association (if Enable is TRUE)

Notes

None

FLASH MANAGER INTERFACE

The flash manager exports a Host interface to allow in−field interrogation, configuration, and programming of the NVM device attached to the AS0147AT. Typically this will be a Flash or EEPROM device.

The capabilities of the flash manager are dependent upon the underlying NVM device driver supported by the firmware. In some cases, a firmware patch must be applied before the flash manager will support its full interface.

OVERVIEW

The NVM resource is shared by a number of components of the AS0147AT firmware; therefore, access to the NVM must be strictly controlled. The Host must obtain an exclusive lock from the flash manager before it can gain access to the SPI device.

FLASH MANAGER HOST COMMANDS

Table 129 summarizes the Host commands relating to the flash manager subsystem of the AS0147AT.

Table 129. FLASH MANAGER HOST COMMANDS

Flash Mgr Host Command	Command	Type	Description
Get Lock	0x8500	Asynchronous	Request the flash manager access lock
Lock Status	0x8501	Synchronous	Retrieve the status of the access lock request
Release Lock	0x8502	Synchronous	Release the flash manager access lock
Config	0x8503	Synchronous	Configure the flash manager and underlying NVM subsystem
Read	0x8504	Asynchronous	Read data from the NVM device
Write	0x8505	Asynchronous	Write data to the NVM device
Erase Block	0x8506	Asynchronous	Erase a block of data from the NVM device
Erase Device	0x8507	Asynchronous	Erase the NVM device
Query Device	0x8508	Asynchronous	Query device−specific information
Flash Status	0x8509	Synchronous	Obtain status of current asynchronous operation
Config Device	0x850A	Synchronous	Configure the attached NVM device
Validate Records	0x850D	Asynchronous	Validate records
Validation Status	0x850E	Synchronous	Obtain status of current asynchronous validate records operation
Issue Device Command	0x850F	Asynchronous	Issue an 8−bit command to an SPI device, with optional transmit payload
Get Device Command Response	0x8510	Synchronous	Retrieve the response to an Issue Device Command request

FLASH MANAGER PARAMETERS

EXTENDED TYPES

Table 130 lists the extended parameter types specific to the flash manager subsystem.

Table 130. FLASH MANAGER PARAMETER TYPES

Extended Type	Size	Field	Base Type	Description
FLASHMGR_CONFIG	48 bits	SPI_SPEED	UINT32	Clock speed of the SPI bus: Min: SPI_SPEED_MIN Max: SPI_SPEED_MAX
		SPI_MODE	UINT8	Mode of the SPI bus: 0: SPI Mode 0 3: SPI Mode 3
		SPI_READ	UINT8	Read mode: 0: Standard (0x03 command) 1: Fast (0x0B command)
DEVICE_CONFIG	64 bits	DRIVER_ID	UINT8	Device driver identifier: 0: SPI_READONLY, mounted by default 1: SPI_FLASH_RW 2: SPI_EEPROM_RW
		READ_CMD	UINT8	Read command: 0: Standard (0x03 command) 1: Fast (0x0B command)
		ADDRESS_WIDTH	UINT8	Address width (in bytes)
		Reserved	UINT8	
		SIZE	UINT32	Size of device (in bytes)

GET LOCK

This asynchronous command requests that the Host obtain the flash manager lock.

Command

Table 131.

Size	Value	Mnemonic	Description
16 bits	0x8500	FLASHMGR_GET_LOCK	Request to obtain the flash manager lock

Command Parameters

None.

Result Status

Table 132.

Result Status Code	Description
ENOERR	Command accepted and in operation
EBUSY	Previous Get Lock request has not completed
EALREADY	Lock has already been obtained.

Command Response Parameters

None.

Notes

- The Host must successfully obtain the flash manager access lock before it can issue any other flash manager commands.
- The Host must ensure the access lock is released when access to the NVM device is no longer required.

LOCK STATUS

Command

This synchronous command retrieves the status of a previous Get Lock request.

Table 133.

Size	Value	Mnemonic	Description
16 bits	0x8501	FLASHMGR_LOCK_STATUS	Get status of a Get Lock request

Command Parameters

None.

Result Status

Table 134.

Result Status Code	Description
ENOERR	Get Lock command has completed – Host now owns the access lock
EBUSY	Previous Get Lock request has not completed

Command Response Parameters

None.

Notes

- The Host can issue further flash manager commands only if this command returns ENOERR.

RELEASE LOCK

This synchronous command releases the flash manager's access lock.

Command

Table 135.

Size	Value	Mnemonic	Description
16 bits	0x8502	FLASHMGR_RELEASE_LOCK	Release the access lock

Command Parameters

None.

Result Status

Table 136.

Result Status Code	Description
ENOERR	Release Lock command has completed
EBUSY	Previous Get Lock request has not completed

Command Response Parameters

None.

Notes

- The Host cannot issue further flash manager commands once the access lock has been released.

CONFIG

This synchronous command configures the flash manager and the underlying SPI support subsystems.

Command

Table 137.

Size	Value	Mnemonic	Description
16 bits	0x8503	FLASHMGR_CONFIG	Configure the flash manager and associated subsystems

Command Parameters

Table 138.

Byte Offset	Field	Type	Value	Description
+0	CONFIG	FLASHMGR_CONFIG		The desired configuration

Result Status

Table 139.

Result Status Code	Description
ENOERR	Configure command has completed
EINVAL	Requested configuration parameter is invalid
EACCES	Host does not own the flash manager access lock

Command Response Parameters

None

Notes

- The SPI_SPEED field of the Config parameter is the desired SPI bus speed in Hertz. The AS0147AT supports a range of SPI bus speeds - see “SPI Bus Speeds” on page 103.
- The AS0147AT uses the following default configuration:
  - SPI _SPEED: Minimum supported (see “SPI Bus Speeds” on page 103)
  - SPI_MODE: Mode 0
  - SPI_READ: Standard 0x03 command
  - The SPI bus speed is dependent upon the current pixel clock setting. If the pixel clock is changed, the SPI bus speed will change proportionally to the change in pixel clock. To avoid a change in SPI bus speed, this command should be re−issued whenever a pixel clock change occurs to reconfigure the SPI bus to the desired speed.

READ

This asynchronous command requests the flash manager to read data from the NVM device.

Command

Table 140.

Size	Value	Mnemonic	Description
16 bits	0x8504	FLASHMGR_READ	Read data from the NVM device

Command Parameters

Table 141.

Byte Offset	Field	Type	Value	Description
+0	Address	UINT32		Address of data to read in NVM
+4	Length	UINT8		Number of bytes to read (limited to PARAMS_POOL_SIZE)

Result Status

Table 142.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
ENODEV	NVM device was not detected
ERANGE	Length is out of range
EBUSY	Previous operation is still in progress
EACCES	Host does not own the flash manager access lock

Command Response Parameters

None

Notes

- The Read command requests the Flash Manager retrieve data from the NVM device. The time to do this is indeterminate.
- The Host should use the Flash Status command to determine if the Read operation has completed.

WRITE

This asynchronous command requests the flash manager write data to NVM device.

Command

Table 143.

Size	Value	Mnemonic	Description
16 bits	0x8505	FLASHMGR_WRITE	Write data to the NVM device

Command Parameters

Table 144.

Byte Offset	Field	Type	Value	Description
+0	Address	UINT32		Address of data to write in NVM
+4	Length	UINT8		Number of bytes to write (limited to PARAMS_POOL_SIZE − 5)
+5	Data[0]	UINT8		First byte of data to write
+n	Data[n−5]	UINT8		Last byte of data to write

Result Status

Table 145.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
ENODEV	NVM device was not detected
ERANGE	Length is out of range
EBUSY	Previous operation is still in progress
ENOSYS	Requested operation is not supported
EACCES	Host does not own the flash manager access lock

Command Response Parameters

None.

Notes

- The Write command requests the Flash Manager write data to the NVM device. The time to do this is indeterminate.
- The Host should use the Flash Status command to determine if the Write operation has completed.
- Where necessary, the Flash Manager will automatically issue a write−enable command to the NVM device before requesting the write operation.
- It is the caller's responsibility to ensure that a write does not span a page boundary in the device being written to.

ERASE BLOCK

This asynchronous command requests the flash manager to erase a block of data in the NVM device.

Command

Table 146.

Size	Value	Mnemonic	Description
16 bits	0x8506	FLASHMGR_ERASE_BLOCK	Erase a block of data in the NVM device

Command Parameters

Table 147.

Byte Offset	Field	Type	Value	Description
+0	Address	UINT32		Address of block of data to erase in NVM

Result Status

Table 148.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
ENODEV	NVM device was not detected
ERANGE	Length is out of range
EBUSY	Previous operation is still in progress
ENOSYS	Requested operation is not supported
EACCES	Host does not own the flash manager access lock

Command Response Parameters

None.

Notes

- The Erase Block command requests the Flash Manager erase a block of data in the NVM device. The time to do this is indeterminate.
- The size of the block to be erased is device−specific and driver−specific.
- The Host should use the Flash Status command to determine if the Erase Block operation has completed.
- Where necessary, the Flash Manager will automatically issue a write−enable command to the NVM device before requesting the erase operation.

ERASE DEVICE

This asynchronous command requests the flash manager to erase the entire NVM device.

Command

Table 149.

Size	Value	Mnemonic	Description
16 bits	0x8507	FLASHMGR_ERASE_DEVICE	Erase NVM device

Command Parameters

None.

Result Status

Table 150.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
ENODEV	NVM device was not detected
EBUSY	Previous operation is still in progress
ENOSYS	Requested operation is not supported
EACCES	Host does not own the flash manager access lock

Command Response Parameters

None.

Notes

- The Erase Device command requests the Flash Manager erase the entire contents of the NVM device. The time to do this is indeterminate.
  - In some circumstances the Erase Device command will operate synchronously – that is, the command will not complete until the device has been erased. This is due to the speed of the SPI bus. Configuring a slower speed will allow the command to operate asynchronously.
- The Host should use the Flash Status command to determine if the Erase Device operation has completed.
- Where necessary, the Flash Manager will automatically issue a write−enable command to the NVM device before requesting the erase operation.

QUERY DEVICE

This asynchronous command requests the flash manager to retrieve the NVM device− and manufacturer−specific identifiers.

Command

Table 151.

Size	Value	Mnemonic	Description
16 bits	0x8508	FLASHMGR_QUERY_DEVICE	Query the NVM device

Command Parameters

None.

Result Status

Table 152.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
ENODEV	NVM device was not detected
EBUSY	Previous operation is still in progress
EACCES	Host does not own the flash manager access lock

Command Response Parameters

None.

Notes

- The Query Device command requests the Flash Manager to retrieve device− and manufacturer−specific information from the NVM device. The time to do this is indeterminate.
- The contents and encoding of the returned data are device− and manufacturer−specific (see “Appendix B: SPI Non−Volatile Memory Device Support” on page 102).
- The Host should use the Flash Status command to determine if the Query Device operation has completed. The Status command also retrieves the response to the Query Device command

FLASH STATUS

This synchronous command retrieves the status of the current flash manager operation.

Command

Table 153.

Size	Value	Mnemonic	Description
16 bits	0x8509	FLASHMGR_STATUS	Retrieve the status of the current operation

Command Parameters

None.

Result Status

Table 154.

Result Status Code	Description
ENOERR	Operation has completed successfully
EIO	Data could not be transferred – operation aborted
EBUSY	Operation is still in progress
EACCES	Host does not own the flash manager access lock

Command Response Parameters

If previous command was Read or Query Device, AND Result Status is ENOERR:

Table 155.

Byte Offset	Field	Type	Value	Description
+0	Data[0]	UINT8		First byte of data
+n	Data[n]	UINT8		Last byte of data

Notes

A result status of ENOERR indicates that the requested operation has completed. In the case of the Read and Query Device command, the Host can now retrieve the requested data from the parameters pool.

CONFIG DEVICE

This synchronous command configures the firmware to control the attached NVM device.

Command

Table 156.

Size	Value	Mnemonic	Description
16−bits	0x850A	FLASHMGR_CONFIG_DEVICE	Configure attached NVM device

Command Parameters

Table 157.

Byte Offset	Field	Type	Value	Description
+0	Config	DEVICE_CONFIG		The desired device configuration

Result Status

Table 158.

Result Status Code	Description
ENOERR	Configure Device command has completed
EINVAL	Requested configuration parameter is invalid
EACCES	Host does not own the Flash Manager access lock

Command Response Parameters

None

Notes

- The AS0147AT firmware cannot erase or write to devices until configured using the Config Device command.
- See “Appendix B: SPI Non−Volatile Memory Device Support” on page 102 for details of the supported NVM devices.
- This command can be encoded within an INIT_TABLE record processed during System Configuration; in this use−case it does not require the Flash Manager access lock to have been obtained using the Get Lock command.

VALIDATE RECORDS

This asynchronous command requests that the firmware search through and validate the correctness of all (or a subset of) the records contained in the attached NVM device.

The command will terminate as soon as an invalid record is found, or when all requested records are parsed and found to be valid.

Command

Table 159.

Size	Value	Mnemonic	Description
16 bits	0x850D	FLASHMGR_VALIDATE_RECORDS	Validates NVM records by iterating through them and comparing their checksums against the computed value

Command Parameters

Table 160.

Byte Offset	Field	Type	Value	Description
+0	Validate Meta Data	FLAG	FALSE TRUE	When FALSE, validates all records referenced by the TOC and its fields. When TRUE, only validates the metadata records. This will reduce the execution time of the command. Note: Data records are validated on− demand when loaded.

Result Status

Table 161.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
ENODEV	NVM device was not detected
ENOMEM	Buffer for record parsing could not be acquired.
EBUSY	Previous operation is still in−progress
EACCES	Could not obtain the Flash Manager access lock

Command Response Parameters

None

Notes

- This command automatically obtains the Flash Manager access lock, and automatically releases it when the command completes (successfully or not). Therefore the Host should not obtain the lock prior to invoking this command. The command will be rejected with EACCES if the lock could not be obtained.
- The command will return ENODEV if an NVM device was not detected during discovery, or if no valid TOC record was detected in the attached device.
- This command can be encoded within an INIT_TABLE record processed during System Configuration to validate the contents of the NVM device before further processing.
- The command attempts to obtain free RAM from the Patch Loader component, to use for temporary storage of the records retrieved from the NVM device. If insufficient free RAM exists (due to patches having been loaded), the command will fail with ENOMEM. The intention is that this command is used prior to loading patches to ensure the NVM device contents are valid; sufficient RAM will be present in this scenario. The free RAM is released back to the Patch Loader when the command completes (successfully or not).
- The Validate Meta Data parameter indicates whether the command should validate only the metadata records present in the NVM device, or all records present (those referenced by the TOC). The NVM records are classed as metadata or data records (refer to Table 162):
  - Metadata records are those which detail where the firmware can locate data records.
  - Data records are those which contain data to be processed by the device firmware or underlying hardware.

Table 162. METADATA AND DATA RECORDS

metadata Records	Data Records
TOC	REGISTER_ARRAY_V1
INIT_TABLE	REGISTER_INIT_V1
PATCH_TOC	REGISTER_SET_V1
OVERLAY_BITMAP_TOC	REGISTER_MASKED_SET_V1
OVERLAY_STRING_TOC	SENSOR_REGISTER_ARRAY_V1
OVERLAY_COLORLUT_TOC	SENSOR_REGISTER_SET_V1
OVERLAY_USERCHAR_TOC	SENSOR_REGISTER_MASKED_SET_V1
COMMAND_SEQ_TOC	VARIABLE_ARRAY_V2
COMMAND_SEQ_V1	VARIABLE_INIT_V2
STE_CONFIG_TOC	VARIABLE_SET_V2
	VARIABLE_MASKED_SET_V2
	PATCH_V1
	OVERLAY_BITMAP_V3
	OVERLAY_STRING_V2
	OVERLAY_COLORLUT
	OVERLAY_USERCHAR
	COMMAND_V1
	STE_CONFIG_V2

- This command does not support the NULL_RECORD or EXTENDED_NULL_RE−CORD records. If such a record is located, the command will terminate. This is because the checksum of these records is invalid, and therefore cannot be used to validate their correctness.
- This command will terminate immediately if an invalid record is detected. Use the Validation Status command to determine the status of the command.
- This command cannot be used to validate the contents of a virtual Flash image. A virtual Flash image is stored in OTPM records which are check summed when read by the firmware during the OTPM−Config state of the System Configuration phase.

VALIDATION STATUS

This synchronous command retrieves the status and results of the previous Validate Records request.

Command

Table 163.

Size	Value	Mnemonic	Description
16 bits	0x850E	FLASHMGR_VALIDATION_STATUS	Retrieves the status of the previous Validate Records request.

Command Parameters

None

Result Status

Table 164.

Result Status Code	Description
ENOERR	Validate Records command completed
EBUSY	Previous Validate Records request has not completed

Command Response Parameters

Table 165.

Byte Offset	Field	Type	Value	Description
+0	Error Status	UINT8	ENOERR – EALREADY	The Result Status of the last error to occur when validating an NVM record
+1	Record Type	UINT8	0x00 – 0xFF	If Error Status is not ENOERR, indicates the type of the record that failed validation
+2	Record Length	UINT16	0x0000 – 0xFFFF	If Error Status is not ENOERR, indicates the payload length of the record that failed validation
+4	Record Checksum	UINT16	0x0000 – 0xFFFF	If Error Status is not ENOERR, indicates the checksum indicated in the header of the record that failed validation
+6	Computed Checksum	UINT16	0x0000 – 0xFFFF	If Error Status is not ENOERR, indicates the checksum value that was computed by the command while the record was being validated.
+8	Failure Address	UINT32	0x00000000 – 0xFFFFFFFF	If Error Status is not ENOERR, indicates the start address in the NVM image of the record that failed validation.
+12	Records Checked	UINT16	0 – 0xFFFF	The total amount of records checked during the last Validate Records command. If Error Status is not ENOERR, this includes the record which failed validation

Notes

The host can issue another Validate Records request only if this command does not return EBUSY.

ISSUE DEVICE COMMAND

This asynchronous command requests the Flash Manager to issue an 8−bit device command to an SPI device.

Command

Table 166.

Size	Value	Mnemonic	Description
16 bits	0x850F	FLASHMGR_ISSUE_DEVICE_CMD	Issue an 8−bit command to an SPI device, with optional transmit payload

Command Parameters

None

Result Status

Table 167.

Byte Offset	Field	Type	Value	Description
+0	Command	UINT8		SPI device command code
+1	Tx Length	UINT8	0 − 253	Number of bytes to write to device following Command byte
+2	Rx Length	UINT8	0 − 256	Number of bytes to read from device following Command byte and the transmit bytes (if Tx Length is non−zero)
+3	Tx Data[0]	UINT8		First byte to write to device following Command byte Ignored if Tx Length = 0
+n	Data[n−3]	UINT8		Last byte of data to write Ignored if Tx Length = 0

Command Response Parameters

Table 168.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
ENODEV	NVM device was not detected
EBADF	NVM device is not an SPI device
ERANGE	Tx Length or Rx Length is out−of−range
EBUSY	Previous operation is still in−progress
ENOSYS	Requested operation is not supported
EACCES	Host does not own the Flash Manager access lock

Notes

- The Issue Device Command command requests the Flash Manager write data to and (optionally) read data from the attached SPI device. The time to do this is indeterminate.
- The Host should use the Get Device Command Response command to determine if the Issue Device Command operation has completed.
- The Command parameter specifies the SPI device command code to be issued. This command code is not interpreted by the Flash Manager.

Examples of Usage

- The Issue Device Command command is only supported by the read−write SPI device drivers. The Host should issue the Config Device command to select the appropriate driver.
- The Issue Device Command and Get Device Command Response host commands can be used to read and write the status register of the attached SPI device. Simplified examples of Read Status Register and Write Status Register are provided below:

```
#define FLASHMGR_ISSUE_DEVICE_COMMAND 0x850F
#define FLASHMGR_GET_DEVICE_CMD_RESPONSE 0x8510

// SPI device commands
#define SPI_READ_STATUS_REG        0x05
#define SPI_WRITE_ENABLE           0x06
#define SPI_WRITE_STATUS_REG       0x01

//-------------------------------------------------------------------------
// issues the FLASHMGR_ISSUE_DEVICE_COMMAND command
// - assumes Host has Flash lock
// - txData is a pointer to an array of bytes (length txLen)
//-------------------------------------------------------------------------
int flashmgrIssueDeviceCommand(uint8 command, uint8 txLen, uint8 rxLen,
    uint8* txData)
{
  int i;
  // set the command and TX length
  writeVar16(CMD_HANDLER_PARAMS_POOL_0, command << 8 | txLen);
  if (0 != txLen) {
   // set the RX length and first byte of TX data
   writeVar16(CMD_HANDLER_PARAMS_POOL_1, rxLen << 8 | txData[0]);

   // write the remaining TX data (ignoring txLen, data will be ignored by firmware anyway)
   for (i=1; i<txLen; i+=2) {
      writeVar16(CMD_HANDLER_PARAMS_POOL_0 + i, txData[i] | txData[i+1]);
}
  } else {
   // set the RX Length
   writeVar16(CMD_HANDLER_PARAMS_POOL_1, rxLen << 8);
  }
  return issueCommand(FLASHMGR_ISSUE_DEVICE_COMMAND);
}

//-------------------------------------------------------------------------
// issues the FLASHMGR_GET_DEVICE_CMD_RESPONSE command
// - assumes Host has Flash lock
// - rxData is an optional pointer to an array of bytes (length rxLen)
//-------------------------------------------------------------------------

int flashmgrGetDeviceCommandResponse(uint8 rxLen, uint8* rxData)
{
  uint16 local_params[8];
  int res = issueCommand(FLASHMGR_GET_DEVICE_CMD_RESPONSE);
  if ((ENOERR == res) && rxLen) {
   // retrieve all data from params pool
   for (i=0; i<rxLen/2; i++) {
      local_params[i] = readVar16(CMD_HANDLER_PARAMS_POOL_0 + (i*2));
   }
   // copy requested data to caller's buffer
   memcpy(rxData, rxLen, (uint8*)local_params);
  }
  return res;
}

//-------------------------------------------------------------------------
// read from the status register of the attached SPI device
// - assumes Host has Flash lock
//-------------------------------------------------------------------------

int readSPIStatusRegister(uint8 *status)
{
```

```
  // Request 1 byte read from the SPI device's status register
  int res = flashmgrIssueDeviceCommand(SPI_READ_STATUS_REG, 0, 1, NULL);
  if (ENOERR == res) {
   // wait for device command to complete
   while (true) {
    res = flashmgrGetDeviceCommandResponse(1, status);
    if (EBUSY != res) break; // we're done
.  }
  }
  return res;
}

//-------------------------------------------------------------------------
// write to the status register of the attached SPI device
// - assumes Host has Flash lock
//-------------------------------------------------------------------------

int writeSPIStatusRegister(uint8 status)
{
  // Must first write-enable the register
  int res = flashmgrIssueDeviceCommand(SPI_WRITE_ENABLE, 0, 0, NULL);
  if (ENOERR == res) {
   // wait for device command to complete
   while (true) {
    res = flashmgrGetDeviceCommandResponse(0, NULL) ;
    if (EBUSY != res) break; // we're done
}
  if (ENOERR == res) {
   // now write 1 byte to the SPI device's status register
   res = flashmgrIssueDeviceCommand(SPI_WRITE_STATUS, 1, 0, &status);
   if (ENOERR == res) {
    // wait for command to complete
    while (true) {
      res = flashmgrGetDeviceCommandResponse(0, NULL);
      if (EBUSY != res) break; // we're done
      }
    }
   }
  }
  return res;
}
```

GET DEVICE COMMAND RESPONSE

This synchronous command retrieves the status and (optional) response data of the last Issue Device Command request.

Command

Table 169.

Size	Value	Mnemonic	Description
16 bits	0x8510	FLASHMGR_GET_DEVICE_CMD_RESP	Retrieve the response to an Issue Device Command request

Command Parameters

None

Result Status

Table 170.

Result Status Code	Description
ENOERR	Operation has completed successfully
EIO	Data could not be transferred – operation aborted
EBUSY	Operation is still in progress
EACCES	Host does not own the Flash Manager access lock

Command Response Parameters

If the Rx Length parameter of the Issue Device Command request was non−zero, AND Result Status is ENOERR:

Table 171.

Byte Offset	Field	Type	Value	Description
+0	Data[0]	UINT8		First byte of Rx data
+n	Data[n]	UINT8		Last byte of Rx data

Notes

A result status of ENOERR indicates that the requested operation has completed. If the Rx Length parameter of the Issue Device Command request was non−zero, the Host can now retrieve the requested data from the Parameters Pool.

Examples of Usage

See“Issue Device Command” example.

SEQUENCER INTERFACE

OVERVIEW

The Sequencer component is responsible for the configuration and management of the AS0147AT sensor and color−pipe, and its associated automatic image processing functions.

Summary

Table 172 summarizes the Host commands relating to the Sequencer subsystem of the AS0147AT.

Table 172. SEQUENCER HOST COMMANDS

Sequencer Host Command	Value	Type	Description
Refresh	0x8606	Asynchronous	Refresh the automatic image processing algorithm configuration
Refresh Status	0x8607	Synchronous	Retrieve the status of the last Refresh operation

Refresh

This asynchronous command instructs the Sequencer to refresh the configuration and state of the automatic image processing algorithms.

Command

Table 173.

Size	Value	Mnemonic	Description
16 bits	0x8606	SEQ_REFRESH	Refresh the automatic image processing algorithm configuration

Command Parameters

None

Result Status

Table 174.

Result Status Code	Description
ENOERR	Command completed successfully

Command Response Parameters

None

Notes

- The Sequencer defers the Refresh operation until the next sensor end−of−frame occurs. The time for this to occur is dependent upon the system state, and the sensor configuration.
- The Host should use the Refresh Status command to determine when the Refresh operation has completed.

REFRESH STATUS

Command

This synchronous command retrieves the status of the last Refresh command.

Table 175.

Size	Value	Mnemonic	Description
16 bits	0x8607	SEQ_REFRESH_STATUS	Retrieve the status of the last Refresh command

Command Parameters

None

Result Status

Table 176.

Result Status Code	Description
ENOERR	Refresh command completed successfully
EBUSY	Refresh command is still in−progress

Command Response Parameters

None

Notes

None

PATCH LOADER INTERFACE

OVERVIEW

The Patch Loader is responsible for loading and applying firmware patches, and for managing the remaining free Patch RAM.

Summary

Table 177 summarizes the Host commands relating to the patch loader subsystem of the AS0147AT.

Table 177. PATCH LOADER HOST COMMANDS

Patch Loader Host Command	Value	Type	Description
Load Patch	0x8700	Asynchronous	Load a patch from NVM and automatically apply
Status	0x8701	Synchronous	Get status of an active Load Patch or Apply Patch request
Apply Patchf	0x8702	Asynchronous	Apply a patch (already located in Patch RAM)
Reserve RAM	0x8706	Synchronous	Reserve RAM to contain a patch

Load Patch

This asynchronous command requests that the firmware loads and applies a firmware patch stored in NVM.

Command

Table 178.

Size	Value	Mnemonic	Description
16 bits	0x8700	PATCHLDR_LOAD_PATCH	Load and apply a patch

Command Parameters

Table 179.

Byte Offset	Field	Type	Value	Description
+0	Patch index	UINT16		Identifies the patch to load

Result Status

Table 180.

Result Status Code	Description
ENOERR	Command accepted and in operation
ENODEV	NVM device was not found – buffer not loaded
ERANGE	Attempting to load patch into already−used area of Patch RAM
EBUSY	Previous Load Patch or Apply Patch request has not completed

Command Response Parameters

None.

Notes

- The Patch ID is determined by the PATCH_TOC record—it is the index of the patch record in the PATCH_TOC.
- If a previous Load Patch command is still active, the request will be rejected with EBUSY.

STATUS

This synchronous command retrieves the status of a previous Load Patch or Apply Patch request.

Table 181.

Size	Value	Mnemonic	Description
16 bits	0x8701	PATCHLDR_STATUS	Get status of a Load Patch or Apply Patch request

Command Parameters

None.

Result Status

Table 182.

Result Status Code	Description
ENOERR	Load Patch or Apply Patch command has completed successfully
ENOENT	Specified patch could not be located in NVM – patch not loaded
EIO	Patch could not be transferred; patch not loaded
EBUSY	Previous Load Patch or Apply Patch request has not completed

Command Response Parameters

None.

Notes

The Host can issue another Load Patch or Apply Patch request if this command does not return EBUSY.

APPLY PATCH

This asynchronous command requests the Patch Loader applies a patch stored in Patch RAM.

Command

Table 183.

Size	Value	Mnemonic	Description
16 bits	0x8702	PATCHLDR_APPLY_PATCH	Apply a patch stored in Patch RAM

Command Parameters

None.

Result Status

Table 184.

Byte Offset	Field	Type	Value	Description
+0	Loader Address	UINT16		Address of the Patch's loader function in Patch RAM
+2	Patch ID	UINT16		Unique patch identifier
+4	Firmware ID	UINT32		Firmware ROM version identifier

Table 185.

Result Status Code	Description
ENOERR	Command accepted and operation in progress
EINVAL	Address is out of range
EBADF	Firmware ID field is invalid
EBUSY	Previous Load Patch or Apply Patch request has not completed

Command Response Parameters

None.

Notes

- The Firmware ID field is used by the Patch Loader to verify that the patch is appropriate for the ROM firmware. Any attempt to load a patch with an incorrect encoded firmware version will be rejected with EBADF.
- The Patch ID field is used to provide a unique identifier for the patch. This is reported by the Patch Loader via the PATCHLDR_PATCH_ID_0 through PATCHLDR_−PATCH_ID_7 variables. No enforcement of unique identifiers is provided by the firmware.
- The patch data loaded into Patch RAM is executable code.

RESERVE RAM

This synchronous command requests the Patch Loader reserve a region of Patch RAM to contain a patch.

Command

Table 186.

Size	Value	Mnemonic	Description
16 bits	0x8706	PATCHLDR_RESERVE_RAM	Request RAM is reserved to contain a patch

Command Parameters

Table 187.

Byte Offset	Field	Type	Value	Description
+0	Start Address	UINT16		Start address of the patch
+2	Size Bytes	UINT16		Size of the patch (in bytes)

Result Status

Table 188.

Result Status Code	Description
ENOERR	Command completed successfully
EBUSY	Insufficient free RAM exists; cannot load patch

Command Response Parameters

None.

Notes

- The Host must successfully reserve the RAM to contain a patch before downloading the patch to Patch RAM.

MISCELLANEOUS

OVERVIEW

The firmware supports a number of miscellaneous commands for processing data stored in NVM, and for synchronization.

Summary

Table 189 summarizes the miscellaneous Host commands.

Table 189. MISCELLANEOUS HOST COMMANDS

Miscellaneous Host Command	Value	Type	Description
Invoke Command Sequence	0x8900	Synchronous	Invokes a sequence of commands stored in NVM
Configure Command Sequence Processor	0x8901	Synchronous	Configures the Command Sequencer processor
Wait for Event	0x8902	Synchronous	Waits for a system event to be signaled

MISCELLANEOUS COMMAND PARAMETERS

Table 190 lists the extended parameter types specific to the miscellaneous commands.

Table 190. MISCELLANEOUS COMMAND PARAMETER TYPES

Extended Type	Size	Field	Base Type	Description
MISC_EVENT_ID	8 bits	–	UINT8	Broadcast event identifiers: 0: Sensor start−of−frame 1: Sensor end−of−frame 2: Overlay start−of−frame 3: Overlay end−of−frame 4:High Temperature 5:Low Temperature 6:Normal Temperature

INVOKE COMMAND SEQUENCE

This synchronous command requests the firmware to invoke (process) a sequence of commands stored in NVM.

Command

Table 191.

Size	Value	Mnemonic	Description
16 bits	0x8900	MISC_INVOKE_COMMAND_SEQ	Process a command sequence stored in NVM

Command Parameters

Table 192.

Byte Offset	Field	Type	Value	Description
+0	Sequence ID	UINT16		Identifies the COMMAND_SEQ_Vx record to invoke

Result Status

Table 193.

Result Status Code	Description
ENOERR	Command completed successfully
ENODEV	NVM device was not found – sequence not invoked
ENOENT	Sequence ID not present – sequence not loaded

Command Response Parameters

None.

Notes

- The Invoke Command Sequence command allows the Host to invoke a Command Sequence (sequence of Host commands) stored within a COMMAND_SEQ_Vx record in NVM.
- The Sequence ID is determined by the COMMAND_SEQ_TOC record - it is the index of the command sequence in the COMMAND_SEQ_TOC.
- The command is synchronous; the Host cannot issue any other commands while the Command Sequence is active. The time to process the Command Sequence is indeterminate.
- Command Sequences can recursively invoke other Command Sequences (by encoding an IInvoke Command Sequence command within a COMMAND_V1 record). The maximum recursion depth is determined by CMDHANDLER_MAX_CMDSEQS.

CONFIGURE COMMAND SEQUENCE PROCESSOR

This synchronous command allows the Host to configure the Command Sequence Processor.

Command

Table 194.

Size	Value	Mnemonic	Description
16 bits	0x8901	MISC_CONFIG_CMDSEQ_PROC	Configures the Command Sequence Processor

Command Parameters

Table 195.

Byte Offset	Field	Type	Value	Description
+0	AbortOnError	FLAG	FALSE TRUE	Continue command sequence processing Abort command sequence processing

Result Status

Table 196.

Result Status Code	Description
ENOERR	Command successful

Command Response Parameters

None.

Notes

- The Command Sequence Processor defaults to aborting processing of command sequences should any command (within the sequence) fail.
- The Host can use this command to configure how the Command Sequencer Processor handles command processing errors.
- If AbortOnError is FALSE, the Command Sequence Processor will attempt to process all commands in the sequence, regardless of each individual command's result status.
- The command sequence will be aborted if the Command Sequence Processor fails to locate or validate any record referenced by the command sequence, regardless of the abort mode.

WAIT FOR EVENT

This synchronous command allows the Host to determine when a broadcast event occurs.

Command

Table 197.

Size	Value	Mnemonic	Description
16 bits	0x8902	MISC_WAIT_FOR_EVENT	Wait for a broadcast event to be signaled

Command Parameters

Table 198.

Byte Offset	Field	Type	Value	Description
+0	EventID	MISC_EVENT_ID		Identifies the event to wait for

Result Status

Table 199.

Result Status Code	Description
ENOERR	Event has been signaled
EINVAL	Invalid event ID
ENOSYS	Command not supported

Command Response Parameters

None.

Notes

- The Wait for Event command blocks the Command Handler task, waiting for the specified broadcast event to be signaled.
- Broadcast events are signaled periodically to indicate the occurrence of a significant system event. The Sequencer signals the sensor end−of−frame event, and the Overlay Manager signals the overlay start− and end−of−frame. The Temperature Monitor, when enabled, signals Temperature High, Low and Normal events.
- The Wait for Event command is synchronous; the Host cannot issue any other commands until the event is signaled. The time to process the command is therefore indeterminate.
- If the component that signals the broadcast event is not active, this command will never complete.
- The Wait for Event command is primarily intended for synchronizing the operations of Command Sequences. For example, a Command Sequence to display an animated bitmap could wait for Overlay Start−Of−Frame events (by encoding the Wait For Event command within a COMMAND_V1 record) between Overlay Load Buffer commands.
- The Wait for Event command also permits the Host to synchronize to the sensor frame timing. The Host issues the Wait For Event command, then continuously polls the Command Register. The command will complete when the sensor start− or end−of− frame event is signaled by the Sequencer.

CALIBRATION STATISTICS

The Calibration Statistics engine is connected to the output of the STE module. In order for it to acquire data the sensor must be streaming, and the system configured for STE− sourced output.

The Calibration Statistics engine has been designed to determine the center of a white cross on a black background. To facilitate this, the module contains accumulators that sum the luma values of every pixel in their bin.

The accumulators can be arranged to sum rows:

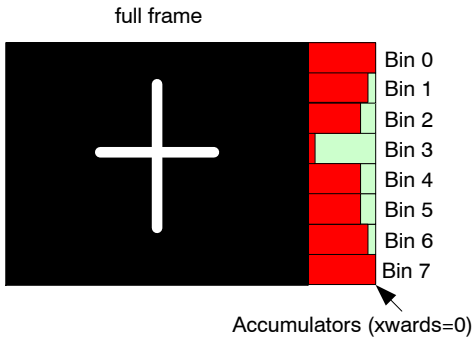

Figure 7.

The accumulators can also be arranged to sum columns:

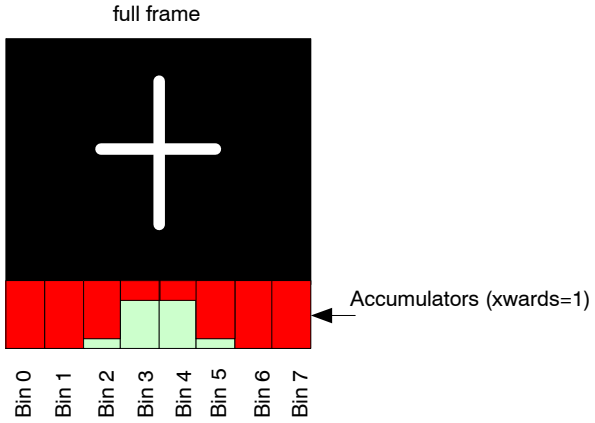

Figure 8.

The diagrams above show the Region Of Interest set to full frame. Using this technique it is possible to get a rough position for the white cross by reading the accumulator results. The Region Of Interest can then be reduced (zoomed in) around this rough position and a better guess achieved. By using this successive approximation technique a number of times a very accurate position can be calculated.

This process is used to automatically find the center of a test target image using the Calib Stats Control command. A single image type is supported: the white cross on dark background.

Summary

Table 200 on page 85 summarizes the host commands that support calibration statistics.

Table 200. CALIBRATION STATISTICS HOST COMMANDS

Calibration Statistics Host Command	Value	Type	Description
Calib Stats Control	0x8B00	Asynchronous	Start statistics gathering
Calib Stats Read	0x8B01	Synchronous	Read the results back

Calibration Statistics Command Parameters

Extended Types

Table 201 lists the extended parameter types specific to the Calibration Statistics commands.

Table 201. CALIBRATION STATISTICS COMMAND PARAMETER TYPES

Extended Type	Size	Field	Base Type	Description
CALIB_OP_TYPE	8 bits	–	UINT8	Calibration statistics operation: 0: Find center of cross−hair target 1: Records the luminance in the specified window

CALIBRATION STATISTICS CONTROL

This asynchronous command configures then enables the Calibration Statistics subsystem to run. The operation performed depends on the parameters passed.

Command

Table 202.

Size	Value	Mnemonic	Description
16 bits	0x8B00	CALIB_STATS_CONTROL	Control the calibration stats subsystem

Command Parameters

Table 203.

Byte Offset	Field	Type	Value	Description
+0	Capture type	CALIB_OP_TYPE		Specifies the operation to perform.
+1	Min_bin_size	UINT8	1 – ROI/8	Minimum bin size (in pixels) in each dimension (when fully zoomed in). Maximum depends on the smaller of the ROI rows or columns divided by 8.
+2	X_start	UINT16		X start of the region of interest in pixels.
+4	X_end	UINT16		X end of the region of interest in pixels.
+6	Y_start	UINT16		Y start of the region of interest in pixels.
+8	Y_end	UINT16		Y end of the region of interest in pixels.
+10	Edge threshold	UINT16	1 – 100	Threshold for chart edge detection between adjacent bins, in percent.
+12	Max angle	UINT16	1 – 7	Maximum rotation angle to tolerate in search, in degrees

Result Status

Table 204.

Result Status Code	Description
ENOERR	Command completed successfully
EBUSY	Previous CALIB_STATS_CONTROL command is still in progress, or the system is not in the SYS_STATE_STREAMING state.

Capture type = 0

If the capture type is “find center of crosshair” the calibration statistics firmware will try to determine the center of a cross in an image. The search window is determined by parameters X_start, X_end, Y_start and Y_end.

The routine starts with this window and gradually zooms the calibration stats window around the brightest bin captured. After a number of iterations, the zoom window reaches maximum resolution Min_bin_size (set by parameters) and the center is recorded.

Where the target cross is on the edge of the search window, it is possible that the returned center will be outside the search window. This is a strong indication that the calibration setup (including the search window and physical position of the camera and cross) are sub−optimal and should be adjusted until the returned center is within the search window.

Capture type = 1

If the capture type is ”record luminance” the calibration statistics firmware will gather data for the specified window, as set by the by parameters X_start, X_end, Y_start and Y_end. The Min_bin_size parameter will be ignored. This window is called the Region Of Interest (ROI) and is divided into 8 bins of equal size.

- The Y bins divide the ROI into 8 rows of equal size.
- The X bins divide the ROI into 8 columns of equal size.

The Y bins are shown below:

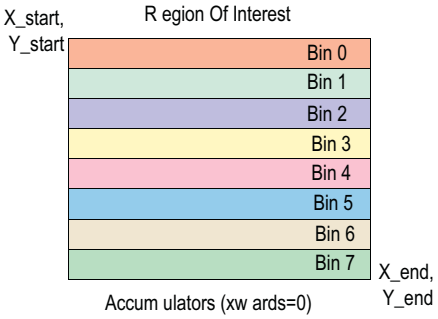

Figure 9.

The luma value of each pixel in a Y bin is summed with the others to produce a 24−bit result.

The X bins are shown below:

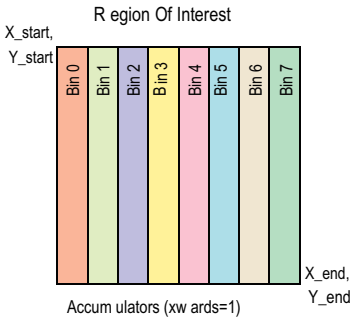

Figure 10.

The luma value of each pixel in an X bin is summed with the others to produce a 24−bit result.

For example, if the ROI is X_start = 0, X_end = 719, Y_start = 0 and Y_end = 479: Each Y bin will contain 43200 pixels:

- Bin 0 (0,0) to (719,59)
- Bin 1 (0,60) to (719,119)
- Bin 2 (0,120) to (719,179)
- Bin 3 (0,180) to (719,239)
- Bin 4 (0,240) to (719,299)
- Bin 5 (0,300) to (719,359)
- Bin 6 (0,360) to (719,419)
- Bin 7 (0,420) to (719,479)

Each X bin will contain 43200 pixels. The X bins will have the following co−ordinates:

- Bin 0 (0,0) to (89,479)
- Bin 1 (90,0) to (179,479)
- Bin 2 (180,0) to (269,479)
- Bin 3 (270,0) to (359,479)
- Bin 4 (360,0) to (449,479)
- Bin 5 (450,0) to (539,479)
- Bin 6 (540,0) to (629,479)
- Bin 7 (630,0) to (719,479)

The summed results are stored to be read by the CALIB_STATS_READ command.

Notes

Before this command is issued the system should be in the SYS_STATE_STREAMING state.

CALIBRATION STATISTICS READ

This synchronous command retrieves the results of a previously−issued Calibration Statistics Control command.

Command

Table 205.

Size	Value	Mnemonic	Description
16 bits	0x8B01	CALIB_STATS_READ	Retrieve results

Command Parameters

Table 206.

Byte Offset	Field	Type	Value	Description
+0	Capture type	CALIB_OP_TYPE		Get the results from the previous Calibration Statistics Control
+1	Bin number	UINT8	0 to 7	Number of accumulator bin to be read. Only required when reading the results of Capture type = 1
+2	Swards	UINT8	1 = X 0 = Y	Rows (Y) or Columns (X). Only required when reading the results of Capture type = 1. A value of 1 selects the column result, otherwise the row results are retrieved.

Result Status

Table 207.

Result Status Code	Description
ENOERR	Command completed successfully
ENOENT	There is no result to read because the requested capture data has not been run or is no longer valid.
ERANGE	The bin number parameter is out of range.
EBUSY	CALIB_STATS_CONTROL is still running or the system is not in the SYS_STATE_STREAMING state.

Command Response Parameters

The response depends upon the Capture type, as indicated in Tables 208 and 209:

Table 208. WHEN CAPTURE TYPE IS SET TO 0

Byte Offset	Field	Type	Value	Description
+0	X_center	UINT16	Pixels	The center of the cross (in pixels)
+2	Y_center	UINT16	Pixels	The center of the cross (in pixels)

Table 209. WHEN CAPTURE TYPE IS SET TO 1

Byte Offset	Field	Type	Value	Description
+0	Sum	UINT32	Luma	This is the sum of the luma values for each pixel in the indexed bin.

Notes

None.

EVENT MONITOR

SUMMARY

The Event Monitor allows command sequences to be associated with a predefined set of system events. The associations are managed using the Host commands shown below and are similar in format to the association commands found in the GPI Monitor component. If an association has been made and enabled a command sequence will be executed when the Event Monitor receives the corresponding system event.

The Event Monitor executes all associations linked to a particular system event, this is done in association order (starting from zero). The Event Monitor also includes an event waiting queue, system events which are received while an event is being handled are placed in the queue to be processed once the current event processing has completed. Queued events are processed in order of arrival; once the waiting queue is full any other events are ignored.

OVERVIEW

Table 210 summarizes the Event Monitor Host commands.

Table 210. EVENT MONITOR HOST COMMANDS

Event Monitor Host Command	Value	Type	Description
Event Monitor Set Association	0x8C00	Synchronous	Associate a system event with a Command Sequence stored in NVM
Event Monitor Get Association	0x8C01	Synchronous	Retrieve an event association

Event Monitor Command Parameters

Extended Types

Table 211 lists the extended parameter types specific to the Event Monitor commands.

Table 211. EVENT MONITOR COMMAND PARAMETER TYPES

Extended Type	Size	Field	Base Type	Description
EVENT_MON_EVENT_ID	8 bits	–	UINT8	Broadcast event identifiers: 0: Sensor start−of−frame 1: Sensor end−of−frame 2: Overlay start−of−frame 3: Overlay end−of−frame 4: High Temperature 5: Low Temperature 6: Normal Temperature

EVENT MONITOR SET ASSOCIATION

This synchronous command instructs the firmware to enable or disable an event association with a Command Sequence stored in NVM.

Command

Table 212.

Size	Value	Mnemonic	Description
16 bits	0x8C00	Event Monitor Set Association	Associate a system event with a Command Sequence stored in NVM

Command Parameters

Table 213.

Byte Offset	Field	Type	Value	Description
+0	Association ID	UINT8	0 .. m−1	Association to modify (where m is EVENT_MONITOR_MAX_ASSOCIATIONS)
+1	Enable	FLAG		TRUE to enable the association, FALSE to disable
+2	Sequence ID	UINT8		Identifies the Command Sequence to load and process
+3	Event ID	EVENT_MON_EVENT_ID		Identifies the System Event that defines this association

Result Status

Table 214.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid association ID
ENOENT	No association exists (if Enable is FALSE)
EBUSY	A recently disabled association is still executing and cannot be modified yet. Try again, or try a different association ID.
EALREADY	Association is already in use.
ERANGE	Invalid event

Command Response Parameters

None

Notes

- The firmware will support EVENT_MONITOR_MAX_ASSOCIATIONS associations between a system event and a Command Sequence.
- The Sequence ID command parameter is not validated by the firmware.
- If a system event occurs which has multiple active command associations, the firm− ware will process these associations in sequential order, beginning from association 0.
- This command is synchronous, in that the association will be enabled or disabled before the command completes. Note however that the Event Monitor task runs asynchronously to the command execution. Due to scheduling priorities, an association may not become active (or inactive) immediately, but will do so within one frame time.

EVENT MONITOR GET ASSOCIATION

This synchronous command returns parameters for the defined association.

Command

Table 215.

Size	Value	Mnemonic	Description
16 bits	0x8C01	Event Monitor Get Association	Get event association parameters

Command Parameters

Table 216.

Byte Offset	Field	Type	Value	Description
+0	Association ID	UINT8	0 .. m−1	Association to be interrogated (where m is EVENT_MONITOR_MAX_ASSOCIATIONS

Result Status

Table 217.

Result Status Code	Description
ENOERR	Command completed successfully
EINVAL	Invalid association ID

Command Response Parameters

Table 218.

Byte Offset	Field	Type	Value	Description
+0	Association ID	UINT8	0 .. m−1	Association to be interrogated (where m is EVENT_MONITOR_MAX_ASSOCIATIONS)
+1	Enabled	FLAG		TRUE if association is enabled, FALSE if disabled
+2	Sequence ID	UINT8		Identifies the Command Sequence to load and process
+3	Event ID	EVENT_MON_EVENT_ID		Identifies the System Event that defines this association

Notes

None

CCI MANAGER

OVERVIEW

Summary

The CCI (camera control interface) Manager exports a Host interface to allow access to the CCI master bus of the AS0147AT.

The CCI master interface is used by the Sensor Manager component of the AS0147AT firmware to control the attached sensor; therefore, access to the CCI bus must be strictly controlled. The Host must obtain an exclusive lock from the CCI Manager before it can gain access to the CCI bus.

When the Host owns the CCI bus lock, the Sensor Manager cannot control the operation of the sensor. This will in turn stall the automatic image processing algorithms. When the Host releases the lock the automatic algorithms will resume.

It is highly recommended that the host only hold the CCI bus lock for the minimum necessary period.

The CCI master interface is provided primarily for sensor configuration at start−up. Its use is discouraged when the system is streaming, due to its affect on the automatic algorithms as described above. During the streaming state, the Sensor Manager will acquire and release the CCI bus lock each frame, in order to update the sensor state. The time that the Sensor Manager holds the lock is indeterminate, and depends upon the changes requested by the automatic algorithms.

If the host needs to read or write to sensor registers while the device is streaming, it should first request the CCI lock using the Get Lock command, then continuously poll the lock status using the Lock Status command. When Lock Status returns ENOERR, then the host should read or write sensor registers as required, and then immediately release the lock with Release Lock. This minimizes the time that the lock will be unavailable to the Sensor Manager.

Table 219 provides a summary of the CCI Manager Host commands. The following subsections detail each command.

Table 219. CCI MANAGER HOST COMMANDS

CCI Manager Host Command	Command	Type	Description
Get Lock	0x8D00	Asynchronous	Request the CCI Manager access lock
Lock Status	0x8D01	Synchronous	Retrieve the status of the access lock request
Release Lock	0x8D02	Synchronous	Release the CCI Manager access lock
Config	0x8D03	Synchronous	Configure the CCI Manager and underlying CCI subsystem
Set Device	0x8D04	Synchronous	Set the target CCI device address
Read	0x8D05	Asynchronous	Read one or more bytes from a 16−bit address
Write	0x8D06	Asynchronous	Write one or more bytes to a 16−bit address
Write Bit−field	0x8D07	Asynchronous	Read−modify−write 16−bit data to a 16−bit address
CCI Status	0x8D08	Synchronous	Obtain status of current asynchronous operation

CCI MANAGER COMMAND PARAMETERS

EXTENDED TYPES

Table 220 lists the extended parameter types specific to the CCI Manager subsystem.

Table 220. CCI MANAGER PARAMETER TYPES

Extended Type	Size	Field	Base Type	Description
CCIMGR_CONFIG	32 bits	CCI_SPEED	UINT32	Clock speed of the CCI bus in Hertz: Min = 100000 Max = 400000

GET LOCK

This asynchronous command requests that the Host obtain the CCI Manager lock.

Command

Table 221.

Size	Value	Mnemonic	Description
16 bits	0x8D00	CCIMGR_GET_LOCK	Request to obtain the CCI Manager lock

Command Parameters

None

Result Status

Table 222.

Result Status Code	Description
ENOERR	Command accepted and in operation
EBUSY	Previous Get Lock request has not completed
EALREADY	Lock has already been obtained

Command Response Parameters

None

Notes

- The Host must successfully obtain the CCI Manager access lock before it can issue any other CCI Manager commands.
- The Host must ensure the access lock is released when access to the CCI bus is no longer required.
- The Host should minimize the time it owns the access lock to prevent stalling the operation of the automatic algorithms of the device.
- The System Manager disables the SYSMGR_SET_STATE Host command when the Host requests the CCIM lock. This prevents the Host requesting system state changes which require the device to control the sensor.
- The System Manager disables the hard standby pin when the Host requests the CCIM lock. This prevents the Host requesting a standby operation which requires the device to control the sensor.

LOCK STATUS

This synchronous command retrieves the status of a previous Get Lock request.

Command

Table 223.

Size	Value	Mnemonic	Description
16 bits	0x8D01	CCIMGR_LOCK_STATUS	Get status of a Get Lock request

Command Parameters

None

Result Status

Table 224.

Result Status Code	Description
ENOERR	Get Lock command has completed – Host now owns the access lock
EBUSY	Previous Get Lock request has not completed

Command Response Parameters

None

Notes

- The Host can issue further CCI Manager commands only if this command returns ENOERR.

RELEASE LOCK

This synchronous command releases the CCI Manager's access lock.

Command

Table 225.

Size	Value	Mnemonic	Description
16 bits	0x8D02	CCIMGR_RELEASE_LOCK	Release the access lock

Command Parameters

None

Result Status

Table 226.

Result Status Code	Description
ENOERR	Release Lock command has completed
EBUSY	Previous Get Lock request has not completed

Command Response Parameters

None

Notes

- The Host cannot issue further CCI Manager commands once the access lock has been released.

CONFIG

This synchronous command configures the CCI Manager and the underlying CCI support subsystems.

Command

Table 227.

Size	Value	Mnemonic	Description
16 bits	0x8D03	CCIMGR_CONFIG	Configure the CCI Manager and associated subsystems

Command Parameters

Table 228.

Byte Offset	Field	Type	Value	Description
+0	Config	CCIMGR_CONFIG		The desired configuration

Result Status

Table 229.

Result Status Code	Description
ENOERR	Configure command has completed
EINVAL	Requested configuration parameter is invalid
EACCES	Host does not own the CCI Manager access lock

Command Response Parameters

None

Notes

- The CCI_SPEED field of the Config parameter is the desired CCI bus speed in Hertz.
- The AS0147AT firmware uses the following default configuration: CCI _SPEED: 400000

SET DEVICE

This synchronous command sets the current CCI device address.

Command

Table 230.

Size	Value	Mnemonic	Description
16 bits	0x8D04	CCIMGR_SET_DEVICE	Set the current CCI device

Command Parameters

Table 231.

Byte Offset	Field	Type	Value	Description
+0	Device Address	UINT8		CCI bus address of device to access, specified as an 8−bit value, where the top 7 bits are the CCI address, and the least−significant is used for read/write indication. A value of 0 indicates that the active sensor device address should be selected (as indicated by CAM_SENSOR_CONTROL_BASE_ADDRESS)

Result Status

Table 232.

Result Status Code	Description
ENOERR	Device address accepted
EBUSY	Previous transaction is still in progress
EACCES	Host does not own the CCI Manager access lock

Command Response Parameters

None

Notes

- The Set Device command sets the CCI device address for all subsequent transfers.

READ

Table 233.

Size	Value	Mnemonic	Description
16 bits	0x8D05	CCIMGR_READ	Read one or more bytes from a 16−bit address

Command

Table 234.

Byte Offset	Field	Type	Value	Description
+0	Register Address	UINT16		Address of CCI device register to read
+2	Length	UINT8	1..122	Number of bytes to read

Command Parameters

This asynchronous command requests the CCI Manager reads one or more bytes from a 16−bit address of a CCI device.

Result Status

Table 235.

Result Status Code	Description
ENOERR	Command accepted and transaction in progress
ERANGE	Length parameter is out of range
EBUSY	Previous transaction is still in−progress
EACCES	Host does not own the CCI Manager access lock

Command Response Parameters

None

Notes

- The Read command requests the CCI Manager perform a CCI transaction to retrieve data from the current CCI device. The time to do this is indeterminate.
- The Host should use the CCI Status command to determine if the Read operation has completed, and to retrieve the read data.

WRITE

This asynchronous command requests the CCI Manager write one or more bytes to a 16−bit address of a CCI device.

Command

Table 236.

Size	Value	Mnemonic	Description
16 bits	0x8D06	CCIMGR_WRITE	Write one or more bytes to a 16−bit address

Command Parameters

Table 237.

Byte Offset	Field	Type	Value	Description
+0	Register Address	UINT16		Address of CCI device register to write
+2	Length	UINT8	1..119	Number of bytes to write

Table 237.

Byte Offset	Field	Type	Value	Description
+3	Data[0]	UINT8		First byte to write
3+n	Data[n]	UINT8		Last byte to write

Result Status

Table 238.

Result Status Code	Description
ENOERR	Command accepted and transaction in progress
ERANGE	Length parameter is out of range
EBUSY	Previous transaction is still in progress
EACCES	Host does not own the CCI Manager access lock

Command Response Parameters

None

Notes

- The Write command requests the CCI Manager perform a CCI transaction to write data to the current CCI device. The time to do this is indeterminate.
- The Host should use the CCI Status command to determine if the Write operation has completed.

WRITE BIT−FIELD

This asynchronous command requests the CCI Manager perform a read−modify−write operation of 16−bit data to a 16−bit address of the current CCI device.

Command

Table 239.

Size	Value	Mnemonic	Description
16 bits	0x8D07	CCIMGR_WRITE_BITFIELD	Read−modify−write data to a CCI device

Command Parameters

Table 240.

Byte Offset	Field	Type	Value	Description
+0	Register Address	UINT16		Address of CCI device register to write
+2	Data	UINT16		Data to write
+4	Mask	UINT16		Mask of data to write

Result Status

Table 241.

Result Status Code	Description
ENOERR	Command accepted and transaction in progress
EBUSY	Previous transaction is still in progress
EACCES	Host does not own the CCI Manager access lock

Command Response Parameters

None

Notes

- The Write Bit−field command requests the CCI Manager perform a CCI transaction to read, modify then write

data to the current CCI device. The time to do this is indeterminate.

- The CCI Manager performs the following operation:

```
val16 = read (device, address)
val16 &= ~mask
val16 |= data
write (device, address, val16)
```

- The Host should use the CCI Status command to determine if the Write Bit−field operation has completed.

CCI STATUS

This synchronous command retrieves the status of the current CCI Manager transaction.

Command

Table 242.

Size	Value	Mnemonic	Description
16 bits	0x8D08	CCIMGR_STATUS	Retrieve the status of the current transaction

None

Result Status

Table 243.

Result Status Code	Description
ENOERR	Transaction has completed successfully
ENOENT	CCI device failed to respond
EIO	Transaction failed − aborted
EBUSY	Transaction is still in progress
EACCES	Host does not own the CCI Manager access lock

Command Response Parameters

If previous transaction was Read, AND Result Status is ENOERR:

Table 244.

Byte Offset	Field	Type	Value	Description
+0	Data[0]	UINT8		First byte read
+n	Data[n]	UINT8		Last byte read

Notes

A result status of ENOERR indicates that the transaction last requested has completed. In the case of the Read command, the Host can now retrieve the requested data from the Parameters Pool.

SENSOR MANAGER

The Sensor Manager exports a Host interface to request discovery and initialization of the sensor attached to the AS0147AT.

OVERVIEW

The Sensor Manager automatically discovers and initializes the sensor attached to the AS0147AT. This occurs during the first Change−Config operation (just prior to entering the streaming state). The Sensor Manager only supports a fixed number of sensors. If the sensor attached is not recognized, the system cannot enter streaming and the Change− Config will be rejected.

The Sensor Manager Host command interface is provided to allow the Host (prior to the first Change−Config) to discover which sensor is attached, and load and apply the appropriate firmware patch, to add support for the attached sensor.

SUMMARY

Table 245 summarizes the Sensor Manager Host commands.

Table 245. SENSOR MANAGER HOST COMMANDS

Sensor Manager Host Command	Value	Type	Description
Discover Sensor	0x8E00	Synchronous	Discover sensor
Initialize Sensor	0x8E01	Synchronous	Initialize attached sensor

Sensor Manager Command Parameters

None

DISCOVER SENSOR HOST COMMAND

This synchronous command requests that the Sensor Manager attempt to discover the sensor attached to the AS0147AT. This command allows a Host to determine the model and revision of the attached sensor, so that the appropriate patches can be loaded prior to initializing it.

Command

Table 246.

Size	Value	Mnemonic	Description
16 bits	0x8E00	SENSOR_MGR_DISCOVER_SENSOR	Discover sensor

Command Parameters

None

Result Status

Table 247.

Result Status Code	Description
ENOERR	Command completed successfully − sensor was detected
EIO	IO error occurred
ENOENT	Failed to detect any sensor
EBUSY	Failed to obtain CCIM lock − retry
EACCES	Cannot perform sensor discovery at this time − retry
EALREADY	The sensor has already been discovered. It can only be performed once.

Command Response Parameters

Table 248.

Byte Offset	Field	Type	Value	Description
+0	CCI Address	UINT8		The CCI address of the detected sensor
+1	Revision	UINT8		The revision of the detected sensor
+2	Model ID	UINT16		The model ID of the detected sensor

Notes

- The sensor will be reset prior to discovery.
- The command will fail with EBUSY if the Host has the CCIM lock.
- The sensor can only be discovered when the system is in the IDLE or SUSPENDED states. The command will fail with EACCES if the system is not in a state that can support discovery.
- The sensor discovery operation can only be performed once following reset or power− up – attempting to issue this command a second time will be rejected with EALREADY.

INITIALIZE SENSOR HOST COMMAND

This synchronous command requests that the Sensor Manager initialize the sensor attached to the AS0147AT. This command allows a Host to manually initialize the sensor prior to the auto−initialization that occurs during the first Config−Change command processing, or at any other time.

Command

Table 249.

Size	Value	Mnemonic	Description
16 bits	0x8E01	SENSOR_MGR_INITIALIZE_SENSOR	Initialize attached sensor

Command Parameters

None

Result Status

Table 250.

Result Status Code	Description
ENOERR	Command completed successfully – sensor initialized
EIO	IO error occurred
ENOENT	No sensor attached
EBADF	Attached sensor is unsupported
EBUSY	Failed to obtain CCIM lock – retry
EACCES	Cannot perform sensor initialization at this time – retry

Command Response Parameters

None

Notes

- The sensor will be initialized. All sensor default records stored in NVM will be applied.
- The command will fail with ENOENT if no sensor is attached, or if the discovery process has not been performed (see“Discover Sensor Host Command” on page 100).
- The command will fail with EBADF if the attached sensor is not supported. The appropriate firmware patch should be loaded and applied, and the command reis−sued.
- The command will fail with EBUSY if the Host has the CCIM lock.
- The sensor can only be initialized when the system is in the IDLE or SUSPENDED states. The command will fail with EACCES if the system is not in a state that can support initialization.
- The sensor initialization operation can only be performed once following reset or power−up – attempting to issue this command a second−time will be rejected with EALREADY.

APPENDIX A: BIG−ENDIAN ENCODING

All 16− and 32−bit data values are encoded in big−endian format - this stores the most− significant data byte at the least−significant (lower) address. For example, the 16−bit value 0x1234 would be stored in memory as:

```
Byte Address Offset +0x0: 0x12 Byte Address
Offset +0x1: 0x34
```

The 32−bit value 0x12345678 would be stored in memory as:

```
Byte Address Offset +0x0: 0x12 Byte Address
Offset +0x1: 0x34 Byte Address Offset +0x2:
0x56 Byte Address Offset +0x3: 0x78
```

APPENDIX B: SPI NON−VOLATILE MEMORY DEVICE SUPPORT

SPI COMMAND SET

The firmware uses the following SPI NVM device commands to implement the Flash Manager host commands. These are detailed in Table 251.

Table 251. SPI NVM DEVICE COMMANDS

SPI Commands	Code	Used for Flash Driver ID	Used for EEPROM Driver ID	Used for Generic Read−only Driver ID
Write Status	0x01	X	X	
Write / Page Program	0x02	X	X	
Read	0x03	X	X	X
Read Status	0x05	X	X	X
Write Enable	0x06	X	X	
Read (fast)	0x0B	X		
Chip Erase	0xC7	X		
Block Erase (64k)	0xD8	X		
Read Manufacturer ID	0xF9	X		X

DEVICE DISCOVERY PROCESS

The process first issues the Read Manufacturer ID SPI NVM command. This is supported by JEDEC−compliant Flash devices. It returns a 3−byte identification code. If this code is non−zero, and also not all 0xFFs, a JEDEC−compliant Flash device is present and discovery completes.

If a JEDEC−compliant Flash device is present the 3−byte identification code is reported in the manuID, deviceID1 and deviceID2 fields as reported by the Query Device command − see “Query Device” on page 65). The Host can use the Query Device command to detect the type of device fitted.

If a JEDEC−compliant Flash device is not present, the firmware then reads 8 bytes from the SPI interface to attempt to discover the presence of an EEPROM device. The firm− ware reads from address 0x0 of the NVM device, looking for the known Version ID field of the TOC record. If this is located an SPI EEPROM is present and discovery completes.

If an NVM device was detected, the Flash Manager automatically mounts the appropriate generic read−only driver for the device (SPI Flash or SPI EEPROM).

PROGRAMMING NVM

The firmware automatically mounts the appropriate generic read−only driver for the attached NVM device during device discovery. If the Host wishes to program the attached NVM device, it must use the Config Device command to select the appropriate read−write driver.

The DRIVER_ID parameter of the Config Device command identifies the type of NVM device that is attached (the assumption being that the Host knows the type and capacity of the attached device). The DRIVER_ID parameter should be set to either:

- 1 if the NVM device is an SPI Flash
- 2 if the NVM device is an SPI EEPROM

NOTE: If the actual capacity of the NVM device is smaller than the SIZE parameter to Config Device, the results of erasing, or writing to, an address outside the device’s capacity are indeterminate.

TYPICAL NVM DEVICE CONFIGURATION VALUES

Table 252 contains typical values for the parameters of the Config Device command for select devices. In this table, 1 kbyte = 1024 bytes; 1 Mbyte = 1024 kbytes = 1024*1024 bytes = 1,048,576 bytes. The SIZE parameter in this table is shown in these convenient units, but the actual command argument is in bytes. All of the listed flash devices support the use of the fast read command, none of the EEPROM devices support the fast read command. That parameter field is not included in this table for brevity.

Table 252. TYPICAL NVM DEVICE CONFIG PARAMETERS

Manufacturer	Device	manuID deviceID1 deviceID2	Type, DRIVER ID Parameter Field	SIZE Parameter Field (bytes)	ADDRESS WIDTH Parameter Field (bytes)
Atmel	AT26DF081A	0x1F4501	Flash, 1	1 Mbyte	3
Atmel	AT25DF161	0x1F4602	Flash, 1	2 Mbyte	3
Sanyo	LE25FW806	0x622662	Flash, 1	1 Mbyte	3
ST	M25P05A	0x202010	Flash, 1	64 kbyte	3
ST	M25P40	0x202013	Flash, 1	512 kbyte	3
ST	M25P80	0x202014	Flash, 1	1 Mbyte	3
ST	M25P16	0x202015	Flash, 1	2 Mbyte	3
ST	M25P32	0x202016	Flash, 1	4 Mbyte	3
ST	M95040	0x20FFFF	EEPROM, 2	512 byte	1
ST	M95020	0x20FFFF	EEPROM, 2	256 byte	1
ST	M95010	0x20FFFF	EEPROM, 2	128 byte	1
ST	M95M01	0x20FFFF	EEPROM, 2	128 kbyte	3
Microchip	M25AA080	0x29FFFF	EEPROM, 2	1 kbyte	2
Microchip	M25LC080	0x29FFFF	EEPROM, 2	1 kbyte	2

SPI BUS SPEEDS

The available SPI bus speeds are dependent upon the pixel clock configuration of the device. The AS0147AT derives the SPI bus clock from the pixel clock, via a configurable clock divider. This can support divisors in the range of 3 through 32. The Flash Manager's Config command will calculate the appropriate divisor to achieve the nearest (less than or equal) value to the requested SPI_SPEED parameter.

APPENDIX C: COMMAND SEQUENCE SUPPORT

The AS0147AT supports a Command Sequence Processor, which allows Host commands to be executed from a command sequence record stored in NVM or OTPM. However, the Processor does not support every command. Unsupported Host commands should not be encoded into command sequence records; their usage within command sequences will cause indeterminate effects.

Note that the Init Table, Patch Init Table, Calibration Table and Overlay Init Table records are also command sequences, except that they are processed automatically by the firm−ware during the System Configuration phase. The same rules apply to these records as to Command Sequence records.

Note that all commands operate synchronously when executed by the Command Sequence processor. When an asynchronous command is executed, the firmware has an internal mechanism to delay the execution of the next command in the sequence until the asynchronous command completes.

Note that some commands have no effect when executed within a command sequence. This is because they return a Result Status, or store response parameters into the Parameters Pool. The Command Sequence Processor cannot operate on this data. Commands with no effect are indicated in Table 43 with an '*'

The Command Sequence Processor also does not support direct writes to the Command register (encoded into REGISTER_SET_V1 records); this will result in undefined behavior. All host commands should be encoded into a COMMAND_V1 record.

The supported and unsupported commands are detailed in Table 253:

Table 253. COMMAND SEQUENCE PROCESSOR COMMAND SUPPORT

Command	Description	Supported by Command Sequence Processor
SYSMGR_SET_STATE	Set the system state	Limited Support. This command cannot be encoded within any record that will be processed during the System Configuration phase (see Appendix G.1). This command can only be encoded with the SYS_STATE_ENTER_CONFIG_CHANGE parameter (see Appendix G.2). Attempting to change the system to any other state will result in indeterminate behavior.
SYSMGR_GET_STATE	Get the system state	Supported*
SYSMGR_CONFIG_POWER_MGMT	Configure power management	Supported
OVRL_ENABLE	Enable overlay	Supported
OVRL_GET_STATE	Get overlay state	Supported*
OVRL_SET_CALIBRATION	Set bitmap/string calibration offset	Supported
OVRL_SET_BITMAP_PROP	Set bitmap property	Supported
OVRL_GET_BITMAP_PROP	Get bitmap property	Supported*
OVRL_SET_STRING0_PROP		
OVRL_SET_STRING1_PROP		
OVRL_SET_STRING2_PROP		
OVRL_SET_STRING3_PROP	Set string property	Supported
OVRL_LOAD_BUFFER	Load buffer (from NVM)	Supported
OVRL_LOAD_STATUS	Status of last load	Supported*
OVRL_WRITE_BUFFER	Write to buffer (via CCI)	Supported. Intended for use via CCI, use OVRL_LOAD_BUFFER for Command Sequences
OVRL_READ_BUFFER	Read buffer (via CCI)	Supported*
OVRL_ENABLE_LAYER	Enable bitmap layer	Supported
OVRL_GET_LAYER_STATUS	Get status of bitmap layer	Supported*
OVRL_SET_STRING0		

Table 253. COMMAND SEQUENCE PROCESSOR COMMAND SUPPORT

Command	Description	Supported by Command Sequence Processor
OVRL_SET_STRING1		
OVRL_SET_STRING2		
OVRL_SET_STRING3	Set string	Supported
OVRL_GET_STRING0		
OVRL_GET_STRING1		
OVRL_GET_STRING2		
OVRL_GET_STRING3	Get string	Supported*
OVRL_LOAD_STRING0		
OVRL_LOAD_STRING1		
OVRL_LOAD_STRING2		
OVRL_LOAD_STRING3	Load string (from NVM)	Supported
OVRL_DRAW_SHAPE	Draw lines and arcs	Supported
OVRL_SET_COLOR_LUT	Set color LUT entries	Supported
OVRL_WRITE_USER_CHAR	Write to User Character RAM	Supported. Intended for use via CCI, use OVRL_LOAD_USER_CHAR for Command Sequences
OVRL_LOAD_COLORLUT	Load color LUT entries	Supported
OVRL_LOAD_USER_CHAR	Load User Character RAM	Supported
STE_CONFIG	Configure from ROM	Supported
STE_LOAD_CONFIG	Load configuration (from NVM)	Supported
STE_LOAD_STATUS	Status of last load	Supported*
STE_WRITE_CONFIG	Write configuration (via CCI)	Supported. Intended for use via CCI, use STE_LOAD_CONFIG for Command Sequences
GPIO_SET_PROP	Set GPIO pin/group property	Supported
GPIO_GET_PROP	Get GPIO pin/group property	Supported*
GPIO_SET_STATE	Set GPIO pin/group state	Supported
GPIO_GET_STATE	Get GPIO pin/group state	Supported*
GPIO_SET_GPI_ASSOC	Set GPI association	Supported
GPIO_GET_GPI_ASSOC	Get GPI association	Supported*
FLASHMGR_GET_LOCK	Get Flash Manager lock	Not Supported. The Command Sequencer Processor obtains the Flash lock in order to retrieve the command sequence record from NVM. Executing this command within a command sequence will result in a dead−lock.
FLASHMGR_LOCK_STATUS	Status of Flash Manager lock	Supported*
FLASHMGR_RELEASE_LOCK	Release Flash Manager lock	Not Supported
FLASHMGR_CONFIG	Configure Flash Manager	Limited Support. This command is only supported during the System Configuration phase (that is, when executed as a result of initialization table processing).
FLASHMGR_READ	Read (from Flash)	Not Supported
FLASHMGR_WRITE	Write (to Flash)	Not Supported
FLASHMGR_ERASE_BLOCK	Erase (Flash) block	Not Supported

Table 253. COMMAND SEQUENCE PROCESSOR COMMAND SUPPORT

Command	Description	Supported by Command Sequence Processor
FLASHMGR_ERASE_DEVICE	Erase (Flash) device	Not Supported
FLASHMGR_QUERY_DEVICE	Query manufacturer and device identifiers	Not Supported
FLASHMGR_STATUS	Status of last Flash operation	Not Supported
FLASHMGR_CONFIG_DEVICE	Configure the SPI device attached	Limited Support. This command is only supported during the System Configuration phase (that is, when executed as a result of initialization table processing).
FLASHMGR_VALIDATE_RECORDS	Validate contents of attached SPI NVM device	Supported
FLASHMGR_VALIDATION_STATUS	Status of the last validate records command	Supported*
FLASHMGR_ISSUE_DEVICE_CMD	Issue low−level SPI device command	Not Supported
FLASHMGR_GET_DEVICE_CMD_RESP	Get response to a low−level SPI device command	Not Supported
SEQ_REFRESH	Refresh Sequencer	Supported
SEQ_REFRESH_STATUS	Retrieve status of last refresh request	Supported*
PATCHLDR_LOAD_PATCH	Load patch (from NVM)	Supported
PATCHLDR_STATUS	Status of last load operation	Supported*
PATCHLDR_APPLY_PATCH	Apply patch	Not Supported. Use PATCHLDR_LOAD_PATCH.
PATCHLDR_RESERVE_RAM	Allocate RAM to contain patch	Not Supported. The PATCHLDR_LOAD_PATCH command performs this operation before loading the patch from NVM.
MISC_INVOKE_COMMAND_SEQ	Invoke a command sequence (stored in NVM)	Supported
MISC_CONFIG_CMDSEQ_PROC	Configure the Command Sequence Processor	Supported
MISC_WAIT_FOR_EVENT	Wait for a System Event	Supported
CALIB_STATS_CONTROL	Configure and control the Calibration Statistics engine	Supported
CALIB_STATS_READ	Retrieve results from the Calibration Statistics engine	Supported*
EVENT_MON_SET_ASSOCIATION	Associate an system event with a Command Sequence stored in NVM	Supported
EVENT_MON_GET_ASSOCIATION	Retrieve an event association	Supported*
CCIMGR_GET_LOCK	Obtain the CCI lock	Supported
CCIMGR_LOCK_STATUS	Status of CCI lock request	Supported*
CCIMGR_RELEASE_LOCK	Release the CCI lock	Supported
CCIMGR_CONFIG	Configure the CCI (master) bus	Supported
CCIMGR_SET_DEVICE	Configure the CCI slave address	Supported
CCIMGR_READ	Read bytes from the CCI bus	Supported*
CCIMGR_WRITE	Write bytes to the CCI bus	Supported
CCIMGR_WRITE_BITFIELD	Read−modify−write to a 16−bit CCI slave register	Supported
CCIMGR_STATUS	Status of current CCI transaction	Supported*

Table 253. COMMAND SEQUENCE PROCESSOR COMMAND SUPPORT

Command	Description	Supported by Command Sequence Processor
SENSOR_MGR_DISCOVER_SENSOR	Discover sensor attached to the AP020X Supported	
SENSOR_MGR_INITIALIZE_SENSOR	Initialize sensor attached to the AP020X Supported	

CHANGING THE SYSTEM STATE DURING THE SYSTEM CONFIGURATION PHASE

At power−up or reset, the device will enter the System Configuration phase, where it determines how the system will be configured. During this time (if so configured) the device will read data from NVM or OTPM.

The NVM or OTPM contains a number of tables of data records termed Init Tables (initialization tables). The device processes each record referenced by an Init Table in turn. The SYSMGR_SET_STATE command is not permitted to be encoded within any records referenced by an Init Table. Note this includes any Command Sequences that could be invoked by the Init Table. Encoding a system state change during the System Configuration phase will result in indeterminate behavior.

To control the next state of the device on completion of the System Configuration phase, set the SYSMGR_CONFIG_MODE firmware variable (encoded within a VARI− ABLE_SET_V2 record). This variable can be set to either:

- SYSMGR_CONFIG_MODE = AUTO_CONFIG (2)
  The system will configure using the default settings.
- SYSMGR_CONFIG_MODE = HOST_CONFIG (3)
  The system will enter SYS_STATE_IDLE (0x20) and wait for further
  Host commands.
- SYSMGR_CONFIG_MODE = CHANGE_CONFIG (4)

The system will apply the current configuration to the hardware, and then start streaming.

The SYSMGR_CONFIG_MODE variable is processed by the firmware at the completion of the System Configuration phase; that is, when all Init Tables have been processed.

Therefore SYSMGR_CONFIG_MODE can be set within any Init Table. If SYSMGR_CON− FIG_MODE is not set, the firmware will go to CONFIG_COMPLETE (5) on completion of the System Configuration phase.

When the System Configuration phase completes, SYSMGR_CONFIG_MODE becomes a read−only variable. The firmware will never read it again.

CHANGING THE SYSTEM STATE FROM COMMAND SEQUENCES

The device enters the Run−time phase immediately on completion of the System Configuration phase. During the Run−time phase, the SYSMGR_SET_STATE command is permitted to be encoded within COMMAND_V1 records referenced from Command Sequences. Note however that only the SYS_STATE_ENTER_CONFIG_CHANGE state−change can be encoded. Encoding a different state change will result in indeterminate behavior.

APPENDIX D: SET STATE COMMAND FAILURE CODES

The System Manager Set State command can be rejected for a variety of reasons. The System Manager provides three status variables to allow the Host to determine why the command was rejected – see Table 255:

Table 254.

System Manager Status Variable	Description
SYSMGR_CMD_STATUS	Indicates the result status of the last Set State command
SYSMGR_CMD_COMP_ID	Indicates which component rejected the command (see Table 255)
SYSMGR_COMD_COMP_FAILURE_ID	Component−specific failure identification code (see tables below)

Table 255. SYSTEM COMPONENT IDENTIFIERS

Component Identifiers	Description
2	Device Manager – see Table 256
5	CAM Control – see Table 257
7	Sensor Manager – see Table 258 on page 109
12	TX Manager – see Table 259 on page 109
15	STE Manager – see Table 260 on page 109
16	RX Manager – see Table 261 on page 109

Table 256. DEVICE MANAGER FAILURE CODES

Device Manager Code	Description
0x0001	Attempting to power off a zone that requires power

Table 257. CAM CONTROL IDENTIFICATION CODES

CAM Control Code	Description
0x0001	Sensor window co−ordinates are invalid
0x0002	Crop window co−ordinates are invalid
0x0003	FOV offsets non−zero when selecting Lens Calibration mode
0x0005	Order of discrete frame rates is invalid (should be high−to−low)
0x0009	Parallel port source selection is invalid
0x000B	CAM mode selection is invalid
0x000C	Failed to claim GPIO pin
0x000D	Synchronization type is unsupported
0x000E	Attempting to leave Synchronized mode to another mode; can only return to Normal mode
0x000F	Attempting to enter Synchronized mode from another mode; can only enter from Normal mode
0x0010	Invalid horizontal crop dimension
0x0011	Invalid vertical crop dimension
0x0013	Either Exposure or White−Balance not in Triggered−Auto mode; both must be in Triggered−Auto mode
0x0014	Output format requires two clocks per pixel, but PLL configuration cannot support this
0x001F	STE transform rotation angle greater than maximum allowed
0x0020	H.264 VBR Luma quality parameter too high for 10−bit encoding
0x0021	H.264 VBR Luma quality parameter too high for 8−bit encoding

Table 258. SENSOR MANAGER FAILURE CODES

Sensor Manager Failure Codes	Description
0x0001	Sensor discovery failed: no sensor attached, or CCI failure
0x0002	Failed to initialize AR0132 sensor: wrong revision
0x0003	Failed to initialize ARX550 sensor: wrong revision
0x0004	Failed to initialize AR0130 sensor: wrong revision
0x0005	Failed to initialize AR0140 sensor: wrong revision
0x0006	Sensor initialization failed: unsupported sensor
0x0007	Sensor initialization failed: CCIM transaction failed
0x0008	Failed to obtain CCIM lock
0x0009	Selected synchronization mode not supported by attached sensor
0x000A	Requested exposure mode not supported by the attached sensor
0x000B	Failed to initialize ASX0340 sensor: wrong revision
0x000C	Failed to initialize AR0230 sensor: wrong revision
0x000D	Failed to initialize AR0231 sensor: wrong revision

Table 259. TX MANAGER FAILURE CODES

TX Manager Code	Description
0x0001	Invalid parallel port source in CAM_PORT_PARALLEL_SOURCE
0x0004	Invalid Bayer path in CAM_OUTPUT_FORMAT_BAYER_PATH
0x0005	Invalid output format in CAM_OUTPUT_FORMAT

Table 260. STE MANAGER FAILURE CODES

STE Manager Code	Description
0x0001	Attached sensor is not supported
0x0003	Configuration cache is invalid
0x0004	Configuration cache is corrupt

Table 261. RX MANAGER FAILURE CODES

RX Manager Code	Description
0x0001	Invalid test pattern selection in CAM_MODE_TEST_PATTERN_SELECT

APPENDIX E: OVERLAY CALIBRATION

The AS0147AT hardware supports position calibration of overlay bitmaps in horizontal and vertical directions; this allows the bitmap position to be adjusted within a +63/−64 pixel range in each direction to allow for manufacturing and fitting tolerances. The calibration offset is applied to all bitmaps whose OVRL_PROP_IS_CALIBRATED property is TRUE.

The default calibration offset is 64 pixels in each direction. However, the calibration offset can be adjusted for each device using the Set Calibration command.

APPENDIX F: HOST COMMAND USAGE EXAMPLE

The following C code shows how a host might implement a Change Config request using the CCI−based host command interface.

```
//-----------------------------------------------------------------------------
// typedefs
//-----------------------------------------------------------------------------
typedef signed char        int8;
typedef unsigned char      uint8;
typedef signed short int   int16;
typedef unsigned short int uint16;
typedef signed long int    int32;
typedef unsigned long int  uint32;
typedef signed long long   int64;
typedef unsigned long long uint64;
typedef signed int         fint;
typedef unsigned int       fuint;
//-----------------------------------------------------------------------------
// Common error codes
//-----------------------------------------------------------------------------

#define  ENOERR     0     //  no error
#define  ENOENT     1     //  no such entity
#define  EINTR      2     //  operation interrupted
#define  EIO        3     //  I/O failure
#define  E2BIG      4     //  too big
#define  EBADF      5     //  bad file/handle
#define  EAGAIN     6     //  would−block, try again
#define  ENOMEM     7     //  not enough memory/resource
#define  EACCES     8     //  permission denied
#define  EBUSY      9     //  entity busy, cannot support operation
#define  EEXIST     10    //  entity exists
#define  ENODEV     11    //  device not found
#define  EINVAL     12    //  invalid argument
#define  ENOSPC     13    //  no space/resource to complete
#define  ERANGE     14    //  parameter out−of−range
#define  ENOSYS     15    //  operation not supported
#define  EALREADY   16    //  already requested/exists

//-----------------------------------------------------------------------------
// Command Handler constants
//-----------------------------------------------------------------------------

#define CMD_HANDLER          31
#define PARAMS_POOL_0        0
#define PARAMS_POOL_1        2
#define PARAMS_POOL_2        4
#define PARAMS_POOL_3        6
#define PARAMS_POOL_4        8
```

```
#define PARAMS_POOL_5       10
#define PARAMS_POOL_6       12
#define PARAMS_POOL_7       14

//-----------------------------------------------------------------------------
// Host Commands
//-----------------------------------------------------------------------------
#define SYSMGR_SET_STATE   0x8100
#define SYSMGR_GET_STATE   0x8101

//-----------------------------------------------------------------------------
// System States
//-----------------------------------------------------------------------------
#define SYS_STATE_ENTER_CONFIG_CHANGE 0x28
#define SYS_STATE_STREAMING         0x31

//-----------------------------------------------------------------------------
// Registers
//-----------------------------------------------------------------------------
#define COMMAND_REG    0x0040
#define COMMAND_REG_DOORBELL_BIT    (1 << 15)

//-----------------------------------------------------------------------------
// The following functions MUST be supplied by the Host, to provide
// read/write access to the CCI registers of the SOC
//-----------------------------------------------------------------------------

// writes val to register reg
extern void writeReg16(uint16 reg, uint16 val);

// returns contents of register reg
extern uint16 readReg16(uint16 reg);

// writes val to 16-bit variable at address var
extern void writeVar16(uint16 drv, uint16 off, uint16 val);

// returns 16-bit variable at address var
extern uint16 readVar16(uint16 var, uint16 off);

// returns 32-bit variable at address var
extern uint32 readVar32(uint16 var, uint16 off);

//-----------------------------------------------------------------------------
// issues a host command, and waits for response
// - returns result status
//-----------------------------------------------------------------------------

int issueCommand(uint16 host_command)
{
  fuint timeout = 100;

   // issue the command
   writeReg16(COMMAND_REG, host_command);

  // and wait for response to confirm command was accepted
  while (0 != timeout)
  {
   uint16 res = readReg16(COMMAND_REG);

   // check if the doorbell bit is clear (0x8000)
```

```
    if ((res & COMMAND_REG_DOORBELL_BIT) == 0)
    {// return the result status
    return (int)res;
   }

    timeout -= 1;
   }

   // device failed to respond
   return EAGAIN;
}

//---------------------------------------------------------------------------
// issues the SYSMGR_SET_STATE command
//---------------------------------------------------------------------------

int setState(uint8 next_state)
{
   // set the desired next state in MSB of params pool 0
   writeVar16(0xFC00 /*CMD_HANDLER_PARAMS_POOL_0*/, next_state << 8);
   return issueCommand(SYSMGR_SET_STATE);
}
//---------------------------------------------------------------------------
// issues the SYSMGR_GET_STATE command
//---------------------------------------------------------------------------
int getState(uint8 *current_state)
{
int res = issueCommand(SYSMGR_GET_STATE);
if ((0 == res) && (NULL != current_state))
{
// retrieve current state from MSB of params pool 0
*current_state = (readVar16(0xFC00 /*CMD_HANDLER_PARAMS_POOL_0*/) >> 8);
}
return res;
}
//---------------------------------------------------------------------------
// requests a Change-Config state change, waits for completion, then
// checks system is streaming
//---------------------------------------------------------------------------

int changeConfig(void)
{
   uint32 timeout = 100;
   uint8 current_state;
   int res = setState(SYS_STATE_ENTER_CONFIG_CHANGE);
   if (ENOERR != res) return res;
   //
   // wait for state change to complete
   while (0 != timeout)
   {
    res = getState(&current_state);
    if (0 == res) break;    // we're done
    if (EBUSY != res) return res;    // something failed
    timeout -= 1;
   }

   if (0 == timeout) return EAGAIN; // host failed to respond in time
   // check the current state
   if (SYS_STATE_STREAMING != current_state) return EINVAL;
   return ENOERR;
}
```

APPENDIX G: CHARACTER ROM

The AS0147AT Overlay character ROM contains the characters in Table 262.

Table 262. CHARACTER ROM

Index (hex)	Unicode code Point	Char	UTF−8 (hex)	Name
0x00	U+0020		0x20	SPACE
0x01	U+0021	!	0x21	EXCLAMATION MARK
0x02	U+0022	“	0x22	QUOTATION MARK
0x03	U+0023	#	0x23	NUMBER SIGN
0x04	U+0024	$	0x24	DOLLAR SIGN
0x05	U+0025	%	0x25	PERCENT SIGN
0x06	U+0026	&	0x26	AMPERSAND
0x07	U+0027	’	0x27	APOSTROPHE
0x08	U+0028	(	0x28	LEFT PARENTHESIS
0x09	U+0029	)	0x29	RIGHT PARENTHESIS
0x0A	U+002A	*	0x2A	ASTERISK
0x0B	U+002B	+	0x2B	PLUS SIGN
0x0C	U+002C	,	0x2C	COMMA
0x0D	U+002D	−	0x2D	HYPHEN−MINUS
0x0E	U+002E	.	0x2E	FULL STOP
0x0F	U+002F	/	0x2F	SOLIDUS
0x10	U+0030	0	0x30	DIGIT ZERO
0x11	U+0031	1	0x31	DIGIT ONE
0x12	U+0032	2	0x32	DIGIT TWO
0x13	U+0033	3	0x33	DIGIT THREE
0x14	U+0034	4	0x34	DIGIT FOUR
0x15	U+0035	5	0x35	DIGIT FIVE
0x16	U+0036	6	0x36	DIGIT SIX
0x17	U+0037	7	0x37	DIGIT SEVEN
0x18	U+0038	8	0x38	DIGIT EIGHT
0x19	U+0039	9	0x39	DIGIT NINE
0x1A	U+003A	:	0x3A	COLON
0x1B	U+003B	;	0x3B	SEMICOLON
0x1C	U+003C	<	0x3C	LESS−THAN SIGN
0x1D	U+003D	=	0x3D	EQUALS SIGN
0x1E	U+003E	>	0x3E	GREATER−THAN SIGN
0x1F	U+003F	?	0x3F	QUESTION MARK
0x20	U+0040	@	0x40	COMMERCIAL AT
0x21	U+0041	A	0x41	LATIN CAPITAL LETTER A
0x22	U+0042	B	0x42	LATIN CAPITAL LETTER B

Table 262. CHARACTER ROM

Index (hex)	Unicode code Point	Char	UTF−8 (hex)	Name
0x23	U+0043	C	0x43	LATIN CAPITAL LETTER C
0x24	U+0044	D	0x44	LATIN CAPITAL LETTER D
0x25	U+0045	E	0x45	LATIN CAPITAL LETTER E
0x26	U+0046	F	0x46	LATIN CAPITAL LETTER F
0x27	U+0047	G	0x47	LATIN CAPITAL LETTER G
0x28	U+0048	H	0x48	LATIN CAPITAL LETTER H
0x29	U+0049	I	0x49	LATIN CAPITAL LETTER I
0x2A	U+004A	J	0x4A	LATIN CAPITAL LETTER J
0x2B	U+004B	K	0x4B	LATIN CAPITAL LETTER K
0x2C	U+004C	L	0x4C	LATIN CAPITAL LETTER L
0x2D	U+004D	M	0x4D	LATIN CAPITAL LETTER M
0x2E	U+004E	N	0x4E	LATIN CAPITAL LETTER N
0x2F	U+004F	O	0x4F	LATIN CAPITAL LETTER O
0x30	U+0050	P	0x50	LATIN CAPITAL LETTER P
0x31	U+0051	Q	0x51	LATIN CAPITAL LETTER Q
0x32	U+0052	R	0x52	LATIN CAPITAL LETTER R
0x33	U+0053	S	0x53	LATIN CAPITAL LETTER S
0x34	U+0054	T	0x54	LATIN CAPITAL LETTER T
0x35	U+0055	U	0x55	LATIN CAPITAL LETTER U
0x36	U+0056	V	0x56	LATIN CAPITAL LETTER V
0x37	U+0057	W	0x57	LATIN CAPITAL LETTER W
0x38	U+0058	X	0x58	LATIN CAPITAL LETTER X
0x39	U+0059	Y	0x59	LATIN CAPITAL LETTER Y
0x3A	U+005A	Z	0x5A	LATIN CAPITAL LETTER Z
0x3B	U+005B	[	0x5B	LEFT SQUARE BRACKET
0x3C	U+005C	\	0x5C	REVERSE SOLIDUS
0x3D	U+005D	]	0x5D	RIGHT SQUARE BRACKET
0x3E	U+005E	^	0x5E	CIRCUMFLEX ACCENT
0x3F	U+005F	_	0x5F	LOW LINE
0x40	U+0060	’	0x60	GRAVE ACCENT
0x41	U+0061	a	0x61	LATIN SMALL LETTER A
0x42	U+0062	b	0x62	LATIN SMALL LETTER B
0x43	U+0063	c	0x63	LATIN SMALL LETTER C
0x44	U+0064	d	0x64	LATIN SMALL LETTER D
0x45	U+0065	e	0x65	LATIN SMALL LETTER E
0x46	U+0066	f	0x66	LATIN SMALL LETTER F
0x47	U+0067	g	0x67	LATIN SMALL LETTER G
0x48	U+0068	h	0x68	LATIN SMALL LETTER H

Table 262. CHARACTER ROM

Index (hex)	Unicode code Point	Char	UTF−8 (hex)	Name
0x49	U+0069	i	0x69	LATIN SMALL LETTER I
0x4A	U+006A	j	0x6A	LATIN SMALL LETTER J
0x4B	U+006B	k	0x6B	LATIN SMALL LETTER K
0x4C	U+006C	l	0x6C	LATIN SMALL LETTER L
0x4D	U+006D	m	0x6D	LATIN SMALL LETTER M
0x4E	U+006E	n	0x6E	LATIN SMALL LETTER N
0x4F	U+006F	o	0x6F	LATIN SMALL LETTER O
0x50	U+0070	p	0x70	LATIN SMALL LETTER P
0x51	U+0071	q	0x71	LATIN SMALL LETTER Q
0x52	U+0072	r	0x72	LATIN SMALL LETTER R
0x53	U+0073	s	0x73	LATIN SMALL LETTER S
0x54	U+0074	t	0x74	LATIN SMALL LETTER T
0x55	U+0075	u	0x75	LATIN SMALL LETTER U
0x56	U+0076	v	0x76	LATIN SMALL LETTER V
0x57	U+0077	w	0x77	LATIN SMALL LETTER W
0x58	U+0078	x	0x78	LATIN SMALL LETTER X
0x59	U+0079	y	0x79	LATIN SMALL LETTER Y
0x5A	U+007A	z	0x7A	LATIN SMALL LETTER Z
0x5B	U+007B	{	0x7B	LEFT CURLY BRACKET
0x5C	U+007C	|	0x7C	VERTICAL LINE
0x5D	U+007D	}	0x7D	RIGHT CURLY BRACKET
0x5E	U+007E	~	0x7E	TILDE
0x5F				EMPTY (UNUSED)
0x60	U+00A0		0xC2A0	NO−BREAK SPACE
0x61	U+00A1	¡	0xC2A1	INVERTED EXCLAMATION MARK
0x62	U+00A2	¢	0xC2A2	CENT SIGN
0x63	U+00A3	£	0xC2A3	POUND SIGN
0x64	U+00A4	¤	0xC2A4	CURRENCY SIGN
0x65	U+00A5	¥	0xC2A5	YEN SIGN
0x66	U+00A6	¦	0xC2A6	BROKEN BAR
0x67	U+00A7	§	0xC2A7	SECTION SIGN
0x68	U+00A8	¨	0xC2A8	DIAERESIS
0x69	U+00A9	©	0xC2A9	COPYRIGHT SIGN
0x6A	U+00AA	ª	0xC2AA	FEMININE ORDINAL INDICATOR
0x6B	U+00AB	”	0xC2AB	LEFT−POINTING DOUBLE ANGLE QUOTATION MARK
0x6C	U+00AC	¬	0xC2AC	NOT SIGN
0x6D	U+00AD		0xC2AD	SOFT HYPHEN
0x6E	U+00AE	®	0xC2AE	REGISTERED SIGN

Table 262. CHARACTER ROM

Index (hex)	Unicode code Point	Char	UTF−8 (hex)	Name
0x6F	U+00AF	¯	0xC2AF	MACRON
0x70	U+00B0	°	0xC2B0	DEGREE SIGN
0x71	U+00B1	±	0xC2B1	PLUS−MINUS SIGN
0x72	U+00B2	²	0xC2B2	SUPERSCRIPT TWO
0x73	U+00B3	³	0xC2B3	SUPERSCRIPT THREE
0x74	U+00B4	´	0xC2B4	ACUTE ACCENT
0x75	U+00B5	µ	0xC2B5	MICRO SIGN
0x76	U+00B6	¶	0xC2B6	PILCROW SIGN
0x77	U+00B7	·	0xC2B7	MIDDLE DOT
0x78	U+00B8	¸	0xC2B8	CEDILLA
0x79	U+00B9	¹	0xC2bB9	SUPERSCRIPT ONE
0x7A	U+00BA	º	0xC2BA	MASCULINE ORDINAL INDICATOR
0x7B	U+00BB	»	0xC2BB	RIGHT−POINTING DOUBLE ANGLE QUOTATION MARK
0x7C	U+00BC	¼	0xC2BC	VULGAR FRACTION ONE QUARTER
0x7D	U+00BD	½	0xC2BD	VULGAR FRACTION ONE HALF
0x7E	U+00BE	¾	0xC2BE	VULGAR FRACTION THREE QUARTERS
0x7F	U+00BF	¿	0xC2BF	INVERTED QUESTION MARK
0x80	U+00C0	À	0xC380	LATIN CAPITAL LETTER A WITH GRAVE
0x81	U+00C1	Á	0xC381	LATIN CAPITAL LETTER A WITH ACUTE
0x82	U+00C2	Â	0xC382	LATIN CAPITAL LETTER A WITH CIRCUMFLEX
0x83	U+00C3	Ã	0xC383	LATIN CAPITAL LETTER A WITH TILDE
0x84	U+00C4	Ä	0xC384	LATIN CAPITAL LETTER A WITH DIAERESIS
0x85	U+00C5	Å	0xC385	LATIN CAPITAL LETTER A WITH RING ABOVE
0x86	U+00C6	Æ	0xC386	LATIN CAPITAL LETTER AE
0x87	U+00C7	Ç	0xC387	LATIN CAPITAL LETTER C WITH CEDILLA
0x88	U+00C8	È	0xC388	LATIN CAPITAL LETTER E WITH GRAVE
0x89	U+00C9	É	0xC389	LATIN CAPITAL LETTER E WITH ACUTE
0x8A	U+00CA	Ê	0xC38A	LATIN CAPITAL LETTER E WITH CIRCUMFLEX
0x8B	U+00CB	Ë	0xC38B	LATIN CAPITAL LETTER E WITH DIAERESIS
0x8C	U+00CC	Ì	0xC38C	LATIN CAPITAL LETTER I WITH GRAVE
0x8D	U+00CD	Í	0xC38D	LATIN CAPITAL LETTER I WITH ACUTE
0x8E	U+00CE	Î	0xC38E	LATIN CAPITAL LETTER I WITH CIRCUMFLEX
0x8F	U+00CF	Ï	0xC38F	LATIN CAPITAL LETTER I WITH DIAERESIS
0x90	U+00D0	Ð	0xC390	LATIN CAPITAL LETTER ETH
0x91	U+00D1	Ñ	0xC391	LATIN CAPITAL LETTER N WITH TILDE
0x92	U+00D2	Ò	0xC392	LATIN CAPITAL LETTER O WITH GRAVE
0x93	U+00D3	Ó	0xC393	LATIN CAPITAL LETTER O WITH ACUTE
0x94	U+00D4	Ô	0xC394	LATIN CAPITAL LETTER O WITH CIRCUMFLEX

Table 262. CHARACTER ROM

Index (hex)	Unicode code Point	Char	UTF−8 (hex)	Name
0x95	U+00D5	Õ	0xC395	LATIN CAPITAL LETTER O WITH TILDE
0x96	U+00D6	Ö	0xC396	LATIN CAPITAL LETTER O WITH DIAERESIS
0x97	U+00D7	×	0xC397	MULTIPLICATION SIGN
0x98	U+00D8	Ø	0xC398	LATIN CAPITAL LETTER O WITH STROKE
0x99	U+00D9	Ù	0xC399	LATIN CAPITAL LETTER U WITH GRAVE
0x9A	U+00DA	Ú	0xC39A	LATIN CAPITAL LETTER U WITH ACUTE
0x9B	U+00DB	Û	0xC39B	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
0x9C	U+00DC	Ü	0xC39C	LATIN CAPITAL LETTER U WITH DIAERESIS
0x9D	U+00DD	Ý	0xC39D	LATIN CAPITAL LETTER Y WITH ACUTE
0x9E	U+00DE	Þ	0xC39E	LATIN CAPITAL LETTER THORN
0x9F	U+00DF	ß	0xC39F	LATIN SMALL LETTER SHARP S
0xA0	U+00E0	à	0xC3A0	LATIN SMALL LETTER A WITH GRAVE
0xA1	U+00E1	á	0xC3A1	LATIN SMALL LETTER A WITH ACUTE
0xA2	U+00E2	â	0xC3A2	LATIN SMALL LETTER A WITH CIRCUMFLEX
0xA3	U+00E3	ã	0xC3A3	LATIN SMALL LETTER A WITH TILDE
0xA4	U+00E4	ä	0xC3A4	LATIN SMALL LETTER A WITH DIAERESIS
0xA5	U+00E5	å	0xC3A5	LATIN SMALL LETTER A WITH RING ABOVE
0xA6	U+00E6	æ	0xC3A6	LATIN SMALL LETTER AE
0xA7	U+00E7	ç	0xC3A7	LATIN SMALL LETTER C WITH CEDILLA
0xA8	U+00E8	è	0xC3A8	LATIN SMALL LETTER E WITH GRAVE
0xA9	U+00E9	é	0xC3A9	LATIN SMALL LETTER E WITH ACUTE
0xAA	U+00EA	ê	0xC3AA	LATIN SMALL LETTER E WITH CIRCUMFLEX
0xAB	U+00EB	ë	0xC3AB	LATIN SMALL LETTER E WITH DIAERESIS
0xAC	U+00EC	ì	0xC3AC	LATIN SMALL LETTER I WITH GRAVE
0xAD	U+00ED	í	0xC3AD	LATIN SMALL LETTER I WITH ACUTE
0xAE	U+00EE	î	0xC3AE	LATIN SMALL LETTER I WITH CIRCUMFLEX
0xAF	U+00EF	ï	0xC3AF	LATIN SMALL LETTER I WITH DIAERESIS
0xB0	U+00F0	ð	0xcC3B0	LATIN SMALL LETTER ETH
0xB1	U+00F1	ñ	0xC3B1	LATIN SMALL LETTER N WITH TILDE
0xB2	U+00F2	ò	0xC3B2	LATIN SMALL LETTER O WITH GRAVE
0xB3	U+00F3	ó	0xC3B3	LATIN SMALL LETTER O WITH ACUTE
0xB4	U+00F4	ô	0xC3B4	LATIN SMALL LETTER O WITH CIRCUMFLEX
0xB5	U+00F5	õ	0xC3B5	LATIN SMALL LETTER O WITH TILDE
0xB6	U+00F6	ö	0xC3B6	LATIN SMALL LETTER O WITH DIAERESIS
0xB7	U+00F7	÷	0xC3B7	DIVISION SIGN
0xB8	U+00F8	ø	0xC3B8	LATIN SMALL LETTER O WITH STROKE
0xB9	U+00F9	ù	0xC3B9	LATIN SMALL LETTER U WITH GRAVE
0xBA	U+00FA	ú	0xC3BA	LATIN SMALL LETTER U WITH ACUTE

Table 262. CHARACTER ROM

Index (hex)	Unicode code Point	Char	UTF−8 (hex)	Name
0xBB	U+00FB	û	0xC3BB	LATIN SMALL LETTER U WITH CIRCUMFLEX
0xBC	U+00FC	ü	0xC3BC	LATIN SMALL LETTER U WITH DIAERESIS
0xBD	U+00FD	ý	0xC3BD	LATIN SMALL LETTER Y WITH ACUTE
0xBE	U+00FE	þ	0xC3BE	LATIN SMALL LETTER THORN
0xBF	U+00FF	ÿ	0xC3BF	LATIN SMALL LETTER Y WITH DIAERESIS

APPENDIX H: CHANGES SINCE AP0100 REV 2

The AS0147AT is based on the AP0100AT. This section describes the changes made for AS0147AT (relative to AP0100AT).

HOST COMMAND CHANGES

The AS0147AT host command set has a number of changes compared to the AP0100AT set – the full AP0100AT command set is summarized in Table 263.

Table 263. AS0147AT HOST COMMANDS VS. AP0100AAT

AP0100 Command	Code	Description	AP020X Implementation
SYSMGR_SET_STATE	0x8100	Set the system state	Supported
SYSMGR_GET_STATE	0x8101	Get the system state	Supported
SYSMGR_CONFIG_POWER_MGMT	0x8102	Configure power management	Supported
OVRL_ENABLE	0x8200	Enable overlay	Supported
OVRL_GET_STATE	0x8201	Get overlay state	Supported
OVRL_SET_CALIBRATION	0x8202	Set bitmap/string calibration offset	Supported
OVRL_SET_BITMAP_PROP	0x8203	Set bitmap property	Supported
OVRL_GET_BITMAP_PROP	0x8204	Get bitmap property	Supported
OVRL_SET_STRING_PROP	0x8205	Set string property	Supported
OVRL_LOAD_BUFFER	0x8206	Load buffer (from NVM)	Supported
OVRL_LOAD_STATUS	0x8207	Status of last load	Supported
OVRL_WRITE_BUFFER	0x8208	Write to buffer (via CCI)	Supported
OVRL_READ_BUFFER	0x8209	Read buffer (via CCI)	Supported
OVRL_ENABLE_LAYER	0x820A	Enable bitmap layer	Supported
OVRL_GET_LAYER_STATUS	0x820B	Get status of bitmap layer	Supported
OVRL_SET_STRING	0x820C	Set string	Supported
OVRL_GET_STRING	0x820D	Get string	Supported
OVRL_LOAD_STRING	0x820E	Load string (from NVM)	Supported
STE_CONFIG	0x8310	Configure (from ROM)	Supported
STE_LOAD_CONFIG	0x8311	Load configuration (from NVM)	Supported
STE_LOAD_STATUS	0x8312	Status of last load	Supported
STE_WRITE_CONFIG	0x8313	Write configuration (via CCI)	Supported
GPIO_SET_PROP	0x8400	Set GPIO pin/group property	Supported
GPIO_GET_PROP	0x8401	Get GPIO pin/group property	Supported
GPIO_SET_STATE	0x8402	Set GPIO pin/group state	Supported
GPIO_GET_STATE	0x8403	Get GPIO pin/group state	Supported
GPIO_SET_GPI_ASSOC	0x8404	Set GPI association	Supported
GPIO_GET_GPI_ASSOC	0x8405	Get GPI association	Supported
FLASHMGR_GET_LOCK	0x8500	Get Flash Manager lock	Supported
FLASHMGR_LOCK_STATUS	0x8501	Status of Flash Manager lock	Supported
FLASHMGR_RELEASE_LOCK	0x8502	Release Flash Manager lock	Supported
FLASHMGR_CONFIG	0x8503	Configure Flash Manager	Supported
FLASHMGR_READ	0x8504	Read (from Flash)	Supported
FLASHMGR_WRITE	0x8505	Write (to Flash)	Supported

Table 263. AS0147AT HOST COMMANDS VS. AP0100AAT

AP0100 Command	Code	Description	AP020X Implementation
FLASHMGR_ERASE_BLOCK	0x8506	Erase (Flash) block	Supported
FLASHMGR_ERASE_DEVICE	0x8507	Erase (Flash) device	Supported
FLASHMGR_QUERY_DEVICE	0x8508	Query manufacturer and device identifiers	Supported
FLASHMGR_STATUS	0x8509	Status of last Flash operation	Supported
FLASHMGR_CONFIG_DEVICE	0x850A	Configure the SPI device attached	Supported
SEQ_REFRESH	0x8606	Refresh Sequencer	Supported
SEQ_REFRESH_STATUS	0x8607	Retrieve status of last refresh request	Supported
PATCHLDR_LOAD_PATCH	0x8700	Load patch (from Flash)	Supported
PATCHLDR_STATUS	0x8701	Status of last load operation	Supported
PATCHLDR_APPLY_PATCH	0x8702	Apply patch	Supported
PATCHLDR_RESERVE_RAM	0x8706	Allocate RAM to contain patch	Supported
MISC_INVOKE_COMMAND_SEQ	0x8900	Invoke a command sequence (stored in Flash	Supported
MISC_CONFIG_CMDSEQ_PROC	0x8901	Configure the Command Sequence Processor	Supported
MISC_WAIT_FOR_EVENT	0x8902	Wait for a System Event	Supported
CALIB_STATS_CONTROL	0x8B00	Configure and control the Calibration Statistics engine	Supported
CALIB_STATS_READ	0x8B01	Retrieve results from the Calibration Statistics engine	Supported
EVENT_MON_SET_ASSOCIATION	0x8C00	Set Event Monitor association	Supported
EVENT_MON_GET_ASSOCIATION	0x8C01	Retrieve details of an Event Monitor association	Supported
CCIMGR_GET_LOCK	0x8D00	Obtain the CCI lock	Supported
CCIMGR_LOCK_STATUS	0x8D01	Status of CCI lock request	Supported
CCIMGR_RELEASE_LOCK	0x8D02	Release the CCI lock	Supported
CCIMGR_CONFIG	0x8D03	Configure the CCI (master) bus	Supported
CCIMGR_SET_DEVICE	0x8D04	Configure the CCI slave address	Supported
CCIMGR_READ	0x8D05	Read bytes from the CCI bus	Supported
CCIMGR_WRITE	0x8D06	Write bytes to the CCI bus	Supported
CCIMGR_WRITE_BITFIELD	0x8D07	Read−modify−write to a 16−bit CCI slave register	Supported
CCIMGR_STATUS	0x8D08	Status of current CCI transaction	Supported
SENSOR_MGR_DISCOVER_SENSOR	0x8E00	Discover sensor attached to the AP020X	Supported
SENSOR_MGR_INITIALIZE_SENSOR	0x8E01	Initialize sensor attached to the AP020X	Supported

OVERLAY MANAGER

The Overlay Manager host command interface has been enhanced to take advantage of the increased capabilities of the AS0147AT:

- Increased the number of overlay layers (from 8 to 12) with 2 each fixed arc and line layers
- Added arc and line drawing commands, with 2 line layers (7, 8) and 2 arc layers (9, 10)
- Each overlay buffer has about 4x the capacity (from 4 kbytes to almost 16 kbytes (16336))
- Each overlay buffer can support more colors (from 16 to 32), so there are more properties and IDs
- The character overlay supports 4 individual strings of 64 characters each
- The overlay character ROM is increased to 192 characters
- New user−defined character RAM provided for up to 64 characters

SYSTEM MANAGER

The state machine of the system configuration phase has been changed.

In AP0100AT, the System Manager checks for Virtual Flash records in OTPM first, and then detects an SPI NVM device and checks for records in it.

In AS0147AT, the System Manager detects an SPI NVM device first. If an SPI NVM device is not detected or it has no valid table of contents record, the System Manager decides whether auto−configuration should be applied or not based on the state of the SPI_DI pin. Then, the System Manager checks for Virtual Flash records in OTPM. If the OTPM does not have valid Virtual Flash records, the System Manager either applies or skips auto−configuration based on the decision made before checking for OTPM Virtual Flash records.

The System Manager detects if records in OTPM or an SPI NVM device attempts to change the SYSMGR_CONFIG_MODE variable value to force the System Manager to switch to the SYS_STATE_FLASH_CONFIG or SYS_STATE_OTPM_CONFIG states. The System Manager blocks that operation and switches to SYS_STATE_IDLE state instead.

The intention is to make the configuration phase deterministic so that loops cannot be created. Precedence is given first to an SPI NVM device, then to OTPM Virtual Flash. The System Manager will not use configuration information from both sources. Any error (invalid data or I/O errors) encountered while in the SYS_STATE_FLASH_CONFIG or SYS_STATE_OTPM_CONFIG states will terminate the configuration phase and the system will be placed in the SYS_STATE_IDLE state.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Email Requests to: orderlit@onsemi.com

ON Semiconductor Website: www.onsemi.com

TECHNICAL SUPPORT
North American Technical Support:
Voice Mail: 1 800−282−9855 Toll Free USA/Canada
Phone: 011 421 33 790 2910

Europe, Middle East and Africa Technical Support:
Phone: 00421 33 790 2910
For additional information, please contact your local Sales Representative