SDK6 AN ADC and IR Input

Version 1.3

September 17, 2014

Confidentiality Notice:

The contents of this document are proprietary and confidential information of Ambarella Inc.

The material in this document is for information only. Ambarella assumes no responsibility for errors or omissions and reserves the right to change, without notice, product specifications, operating characteristics, packaging, ordering, etc. Ambarella assumes no liability for damage resulting from the use of information contained in this document.

US
3101 Jay Street
Ste. 110
Santa Clara, CA 95054, USA
Phone: +1.408.734.8888
Fax: +1.408.734.0788

Taiwan
Suite C1, No. 1, Li-Hsin Road 1
Science-Based Industrial Park
Hsinchu 30078, Taiwan
Phone: +886.3.666.8828
Fax: +886.3.666.1282

Hong Kong
Unit A&B, 18/F, Spectrum Tower
53 Hung To Road
Kwun Tong, Kowloon
Phone: +85.2.2806.8711
Fax: +85.2.2806.8722

Japan - Yokohama
Shin-Yokohama Business Center Bldg. 5th Floor
3-2-6 Shin-Yokohama, Kohoku-ku,
Yokohama, Kanagawa, 222-0033, Japan
Phone: +81.45.548.6150
Fax: +81.45.548.6151

Korea
6 Floor, Hanwon-Bldg.
Sunae-Dong, 6-1, Bundang-Gu
SeongNam-City, Kyunggi-Do
Republic of Korea 463-825
Phone: +031.717.2780
Fax: +031.717.2782

China - Shenzhen
Unit E, 5th Floor
No. 2 Finance Base
8 Ke Fa Road
Shenzhen, 518057, China
Phone: +86.755.3301.0366
Fax: +86.755.3301.0966

China - Shanghai
9th Floor, Park Center
1088 Fangdian Road, Pudong New District
Shanghai 201204, China
Phone: +86.21.6088.0608
Fax: +86.21.6088.0366

I Contents

II Preface

This document provides technical details using a set of consistent typographical conventions to help the user differentiate key concepts at a glance.

Conventions include:

Example	Description
AmbaGuiGen, **DirectUSB** **Save, File > Save** **Power, Reset, Home**	Software names GUI commands and command sequences Computer / Hardware buttons
Flash_IO_control **da**, **status**, **enable**	Register names and register fields. For example, **Flash_IO_control** is the register for global control of Flash I/O, and bit 17 (**da)** is used for DMA acknowledgement.
GPIO81, **CLK_AU**	Hardware external pins
VIL, VIH, VOL, VOH	Hardware pin parameters
INT_O, RXDATA_I	Hardware pin signals
AmbaI2C_Init() **AMBA_I2C_CTRL_s** **AMBA_I2C_CHANNEL_e** **AMBA_GIC_ISR_f** **AMBA_KAL_TASK_t**	API Functions API Structures API Enumerations API Function pointers API Typedef of ThreadX kernel abstraction layer

`DSC_Platform\Tools`
`AmbaI2C.h`
`RetStatus = AmbaI2C_Init();`
	User entries into software dialogues and GUI windows
	File names and paths
	Command line scripting and Code

Table II-1. Typographical Conventions for Technical Documents.

Additional Ambarella typographical conventions include:

- Acronyms are given in UPPER CASE using the default font (e.g., AHB, ARM11 and DDRIO).
- Names of Ambarella documents and publicly available standards, specifications, and databooks appear in *italic* type.

1 Overview

1.1 Overview: Introduction

This Application Note (AN) describes the Key-Scan Application Programming Interface (API) from Ambarella. This API is used to detect button-press events and implement key event handlers for the Analog to Digital (ADC) and InfraRed (IR) Remote interfaces.

The document is organized as follows:

- (Chapter 2) ADC Key Scan API
- (Chapter 3) IR Remote Key Scan API

1.2 Overview: Key Scan Events

The Ambarella development platform supports two input methods. The first input method is a button controlled by the **ADC key** on the main board. The second input method is the InfraRed (IR) Remote Controller key (**IR key**). When a key on either interface is pressed, the event is detected by either the ADC key-scan or IR key-scan API. These APIs allow the implementation of multiple key event handlers.

1.3 Overview: Scope of Document

The APIs described in this document are intended to provide an experienced programmer with the background required to develop and implement ADC and IR key-scan input in a production product. It is assumed that the reader of this document is in possession of a development board for testing and is familiar with the Ambarella chip hardware and system software. For additional background, please refer to the following documents:

- The chip datasheet provides hardware pin and package details including a feature list with descriptions of chip performance, brief interface descriptions, a complete power-on configuration table and electrical characteristics.

2 ADC Key Scan API

2.1 ADC Key Scan: Overview

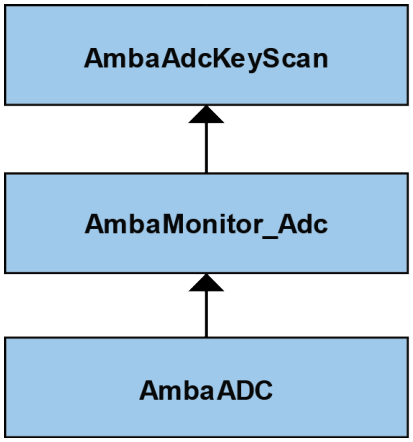
This chapter describes the ADC Key-Scan API as follows:

- (Section 2.2) ADC Key Scan: Background
- (Section 2.3) ADC Key Scan: List of Functions

2.2 ADC Key Scan: Background

The Ambarella system on a chip (SoC) provides twelve 12-bit Analog-to-Digital Conversion channels for low-speed monitoring functions. The ADC interface includes a threshold control feature, allowing interrupts to be sent to the Vector Interrupt Controller (VIC) if the sampled voltage exceeds a specified maximum or minimum threshold value. This capability enables the software to use an interrupt scheme to monitor voltage changes rather than continuously polling ADC channel voltages. The ADC can handle inputs at 0 - 3.3 V.

ADC Channel 1 and ADC Channel 3 have been designated for use with buttons on the main board. **AmbaMonitor_Adc** is used to send the ADC data, while **AmbaAdcKeyScan** specifies the current key code.



Define multiple ranges for the ADC channel.
Give event on key changes.

Periodic monitoring.
Give ADC values of each channels.

Middleware

Table 2-1. ADC Key Scan Process.

The ADC Key-Scan API enables the following functions to be performed:

- Initializing a Monitoring Task
- Registering a Key Event
- De-Registering a Key Event
- Setting the Monitoring Time Period
- Initializing an ADC Key Scan
- Adding an ADC Key Event
- Removing a Key Event

2.3 ADC Key Scan: List of Functions

- AmbaMonitor_AdcInit
- AmbaMonitor_AdcRegisterEventHandler
- AmbaMonitor_AdcUnRegisterEventHandler
- AmbaMonitor_AdcSetPeriod
- AmbaAdcKeyScan_Init
- AmbaAdcKeyScan_RegisterKeyEventHandler
- AmbaAdcKeyScan_UnRegisterKeyEventHandler

2.3.1 AmbaMonitor_AdcInit

ADC Key Scan

API Syntax:

int **AmbaMonitor_AdcInit** (UINT32 TaskPriority, UINT32 MonitorPeriod)

Function Description:

- This function is used to initialize an ADC monitoring task. Several initialization functions are included: **AmbaADC_Init()**, **AmbaMonitor_AdcInit()** and **AmbaAdcKeyScan_Init()**. Note that all of these initialization functions must be called in order to perform a successful key-scan.

Parameters:

Type	Parameter	Description
UINT32	**TaskPriority**	Priority of the collection task
UINT32	**MonitorPeriod**	Time period over which ADC data is to be collected (milliseconds)

*Table 2-2. Parameters for ADC IR API **AmbaMonitor_AdcInit()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 2-3. Returns for ADC IR API **AmbaMonitor_AdcInit()**.*

Example:

```
/* Initialize ADC key scan */
AmbaADC_Init();
AmbaMonitor_AdcInit(ADC_TASK_PRIORITY, 10);
AmbaAdcKeyScan_Init(sizeof(AdckeyScanHandler), AdcKeyScanHandler);
```

See Also:

AmbaAdcKeyScan_Init()

2.3.2 AmbaMonitor_AdcRegisterEventHandler

API Syntax:

int **AmbaMonitor_AdcRegisterEventHandler** (AMBA_ADC_CHANNEL_e AdcChanID, AMBA_ADC_EVENT_HANDLER_f EventHandler)

Function Description:

- This function is used to register an Event Handler. Any given ADC channel is only permitted to have one associated event handler. The event handler for a given ADC channel is registered using **AMBA_ADC_EVENT_HANDLER_f**. One event can, however, be registered with two or more ADC channels. For example, the **AdcCh1andCh3EventHandler** function is registered to ADC channels one and three.
- By default, this function is used by **AmbaAdcKeyScan_Init()** to set the ADC Channel 1 and Channel 3 event handlers.

Parameters:

Type	Parameter	Description
AMBA_ADC_CHANNEL_e	**AdcChanID**	ADC channel number
AMBA_ADC_EVENT_HANDLER_f	**EventHandler**	Pointer to the Event Handler `typedef void (*AMBA_ADC_EVENT_HANDLER_f)(int, UINT16);`

*Table 2-4. Parameters for ADC IR API **AmbaMonitor_AdcRegisterEventHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 2-5. Returns for ADC IR API **AmbaMonitor_AdcRegisterEventHandler()**.*

Example:

```
AmbaMonitor_AdcRegisterEventHandler(AMBA_ADC_CHANNEL1,
AdcCh1andCh3EventHandler);
AmbaMonitor_AdcRegisterEventHandler(AMBA_ADC_CHANNEL3,
AdcCh1andCh3EventHandler);
```

See Also:

AmbaMonitor_AdcUnRegisterEventHandler()
AmbaAdcKeyScan_Init()

2.3.3 AmbaMonitor_AdcUnRegisterEventHandler

API Syntax:

int **AmbaMonitor_AdcUnRegisterEventHandler** (AMBA_ADC_CHANNEL_e AdcChanID)

Function Description:

- This function is used to de-register an Event Handler. This function should be used to de-register an existing event handler prior to assigning a new one.

Parameters:

Type	Parameter	Description
AMBA_ADC_CHANNEL_e	**AdcChanID**	ADC channel number

*Table 2-6. Parameters for ADC IR API **AmbaMonitor_AdcUnRegisterEventHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 2-7. Returns for ADC IR API **AmbaMonitor_AdcUnRegisterEventHandler()**.*

Example:

```
AmbaMonitor_AdcUnRegisterEventHandler(AMBA_ADC_CHANNEL1);
AmbaMonitor_AdcUnRegisterEventHandler(AMBA_ADC_CHANNEL3);;
```

See Also:

AmbaMonitor_AdcRegisterEventHandler()

2.3.4 AmbaMonitor_AdcSetPeriod

API Syntax:

int **AmbaMonitor_AdcSetPeriod** (UINT32 Period)

Function Description:

- This function is used to specify the time period over which ADC values are to be monitored.
- The default time period value is set by the **AmbaMonitor_Init()** function.

Parameters:

Type	Parameter	Description
UINT32	**Period**	Time period over which ADC values are to be monitored (milliseconds)

*Table 2-8. Parameters for ADC IR API **AmbaMonitor_AdcSetPeriod()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 2-9. Returns for ADC IR API **AmbaMonitor_AdcSetPeriod()**.*

Example:

```
/* Set ADC Monitoring Period */
AmbaMonitor_AdcSetPeriod(20); /* Set as 20ms*/
```

See Also:

AmbaMonitor_AdcInit

2.3.5 AmbaAdcKeyScan_Init

API Syntax:

int **AmbaAdcKeyScan_Init** (int MaxNumHandlers, AMBA_KEY_SCAN_EVENT_HANDLER_f *pEventHandlers)

Function Description:

- This function is used to initialize an ADC key-scan. The **AmbaMonitor_AdcInit** function must be called prior to calling this function.
- Both the maximum number of handlers and the first event handler must be specified.

Parameters:

Type	Parameter	Description
int	**MaxNumHandlers**	Maximum number of Handlers
AMBA_KEY_SCAN_EVENT_HANDLER_f*	**pEventHandlers**	Pointer to the Event Handlers `typedef void (*AMBA_KEY_SCAN_EVENT_HANDLER_f)(UINT16 KeyCode);`

*Table 2-10. Parameters for ADC IR API **AmbaAdcKeyScan_Init()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 2-11. Returns for ADC IR API **AmbaAdcKeyScan_Init()**.*

Example:

```
/* Initialize ADC key scan */
AmbaADC_Init();
AmbaMonitor_AdcInit(ADC_TASK_PRIORITY, 10);
AmbaAdcKeyScan_Init(sizeof(AdckeyScanHandler), AdcKeyScanHandler);
```

See Also:

AmbaAdcMonitor_AdcInit()

2.3.6 AmbaAdcKeyScan_RegisterKeyEventHandler

API Syntax:

int **AmbaAdcKeyScan_RegisterKeyEventHandler** (AMBA_KEY_SCAN_EVENT_HANDLER_f Handler)

Function Description:

- This function is used to add an ADC key event handler. When a system detects an ADC button-press, all registered event handlers will be called.

Parameters:

Type	Parameter	Description
AMBA_KEY_SCAN_EVENT_HANDLER_f	**Handler**	Pointer to the Key Event Handler `typedef void (*AMBA_KEY_SCAN_EVENT_HANDLER_f)(UINT16 KeyCode);`

*Table 2-12. Parameters for ADC IR API **AmbaAdcKeyScan_RegisterKeyEventHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 2-13. Returns for ADC IR API **AmbaAdcKeyScan_RegisterKeyEventHandler()**.*

Example:

```
/* Register KeyScanEvent */
AmbaAdcKeyScan_RegisterKeyEventHandler(KeyScanEvent);)
```

See Also:

AmbaAdcKeyScan_UnRegisterKeyEventHandler()

2.3.7 AmbaAdcKeyScan_UnRegisterKeyEventHandler

API Syntax:

AmbaAdcKeyScan_UnRegisterKeyEventHandler (AMBA_KEY_SCAN_EVENT_HANDLER_f Handler)

Function Description:

- This function is used to remove an ADC key event handler.

Parameters:

Type	Parameter	Description
AMBA_KEY_SCAN_EVENT_HANDLER_f	**Handler**	Pointer to Key Event Handler `typedef void (*AMBA_KEY_SCAN_EVENT_HANDLER_f)(UINT16 KeyCode);`

*Table 2-14. Parameters for ADC IR API **AmbaAdcKeyScan_UnRegisterKeyEventHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 2-15. Returns for ADC IR API **AmbaAdcKeyScan_UnRegisterKeyEventHandler()**.*

Example:

```
/* UnRegister KeyScanEvent */
AmbaAdcKeyScan_UnRegisterKeyEventHandler(KeyScanEvent);
```

See Also:

AmbaAdcKeyScan_RegisterKeyEventHandler()

3 IR Remote Key Scan API

3.1 IR Remote Key Scan: Overview

This chapter describes the IR Remote Key-Scan API as follows:

- (Section 3.2) IR Remote Key Scan: Background
- (Section 3.3) IR Remote Key Scan: List of Functions

3.2 IR Remote Key Scan: Background

The Ambarella SoC provides an interface that handles digital signals from IR Remote controllers. There are a number of IR remote control protocols used by various consumer electronics manufacturers, most of which are variants of the **RC-5** and **REC-80** standards. A remote command consists of a fixed-length binary code word. Each bit of a code word is encoded using either bi-phase coding (**RC-5**) or pulse-length coding (**REC-80**)

The **AmbaIrKeyScan_Init** function registers a Key Scan Handler to **AmbaMonitor_IrRemote**. It also parses key code and provides key events.

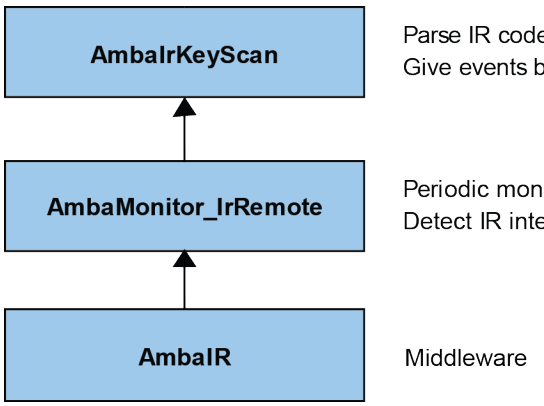


Table 3-1. IR Remote Key Scan Process.

The IR Remote Key-Scan API enables the following functions to be performed:

- Initializing a Monitoring Task
- Registering a Key Event
- De-Registering a Key Event
- Initializing an IR Remote Key Scan
- Adding a Key Event
- Removing a Key Event

3.3 IR Remote Key Scan: List of Functions

- AmbaMonitor_IrRemoteInit
- AmbaMonitor_IrRemoteRegisterEventHandler
- AmbaMonitor_IrRemoteUnRegisterHandler
- AmbaIrKeyScan_Init
- AmbaIrKeyScan_RegisterKeyEventHandler
- AmbaIrKeyScan_UnRegisterKeyEventHandler

3.3.1 AmbaMonitor_IrRemoteInit

API Syntax:

int **AmbaMonitor_IrRemoteInit** (UINT32 TaskPriority)

Function Description:

- This function is used to initialize the IR monitor. Several initialization functions are included: **AmbaIR_Init()**, **AmbaMonitor_IrRemoteInit()** and **AmbaIrKeyScan_Init()**. Note that all of these initialization functions must be called in order to perform a successful key-scan.

Parameters:

Type	Parameter	Description
UINT32	**TaskPriority**	Priority of the collection task

*Table 3-2. Parameters for ADC IR API **AmbaMonitor_IrRemoteInit()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 3-3. Returns for ADC IR API **AmbaMonitor_IrRemoteInit()**.*

Example:

```
/* Initialize IR key scan */
AmbaIR_Init();
AmbaMonitor_IrRemoteInit(IR_TASK_PRIORITY);
AmbaIrKeyScan_Init(sizeof(IrKeyScanHandlers), IrKeyScanHandlers);
```

See Also:

AmbaIrKeyScan_Init()

3.3.2 AmbaMonitor_IrRemoteRegisterEventHandler

API Syntax:

int **AmbaMonitor_IrRemoteRegisterEventHandler** (AMBA_IR_EVENT_HANDLER_f Handler)

Function Description:

- This function is used to add an IR monitoring event. IR remote monitoring can only be associated with one event handler. The event handler is registered using **AMBA_IR_EVENT_HANDLER_f**.
- By default, the API **AmbaIrKeyScan_Init()** uses this function to set the event handler for IR buttons.

Parameters:

Type	Parameter	Description
AMBA_IR_EVENT_HANDLER_f	**Handler**	IR monitor handler `typedef void (*AMBA_IR_EVENT_HANDLER_f) (UINT32, UINT16 *);`

*Table 3-4. Parameters for ADC IR API **AmbaMonitor_IrRemoteRegisterEventHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 3-5. Returns for ADC IR API **AmbaMonitor_IrRemoteRegisterEventHandler()**.*

Example:

```
AmbaMonitor_IrRemoteRegisterEventHandler(IrEventHandler);
```

See Also:

AmbaMonitor_IrRemoteUnRegisterHandler()
AmbaIrKeyScan_Init()

3.3.3 AmbaMonitor_IrRemoteUnRegisterHandler

API Syntax:

int **AmbaMonitor_IrRemoteUnRegisterHandler** (AMBA_IR_EVENT_HANDLER_f Handler)

Function Description:

- This function is used to delete an IR monitoring event. This function should be used to de-register an existing event handler prior to assigning a new one.

Parameters:

Type	Parameter	Description
AMBA_IR_EVENT_HANDLER_f	**Handler**	IR monitor handler `typedef void (*AMBA_IR_EVENT_HANDLER_f) (UINT32, UINT16 *);`

*Table 3-6. Parameters for ADC IR API **AmbaMonitor_IrRemoteUnRegisterHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 3-7. Returns for ADC IR API **AmbaMonitor_IrRemoteUnRegisterHandler()**.*

Example:

```
AmbaMonitor_IrRemoteUnRegisterEventHandler(IrEventHandler);
```

See Also:

AmbaMonitor_IrRemoteRegisterHandler()

3.3.4 AmbaIrKeyScan_Init

API Syntax:

int **AmbaIrKeyScan_Init** (int MaxNumHandlers, AMBA_IR_KEY_EVENT_HANDLER_f *pEventHandlersq)

Function Description:

- This function is used to initialize an IR Remote Key Scan. The **AmbaMonitor_IrRemoteInit** function must be called prior to calling this function.
- Both the maximum number of handlers and the first event handler must be specified.

Parameters:

Type	Parameter	Description
AMBA_IR_PROTOCOL_e	**IrProtocol**	IR Protocol type
int	**MaxNumHandlers**	Maximum number of Handlers
AMBA_IR_KEY_EVENT_HANDLER_f*	**pEventHandlers**	Pointer to the Event Handlers `typedef void (*AMBA_IR_KEY_EVENT_HANDLER_f)(UINT16 KeyCode);`

*Table 3-8. Parameters for ADC IR API **AmbaIrKeyScan_Init()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 3-9. Returns for ADC IR API **AmbaIrKeyScan_Init()**.*

Example:

```
/* Initialize ADC key scan */
AmbaIR_Init();
AmbaMonitor_IrRemoteInit(IR_TASK_PRIORITY);
AmbaIrKeyScan_Init(AMBA_IR_PROTOCOL_PANASONIC,
sizeof(IrKeyScanHandlers), IrKeyScanHandlers);
```

See Also:

AmbaMonitor_IrRemoteInit()

3.3.5 AmbaIrKeyScan_RegisterKeyEventHandler

API Syntax:

int **AmbaIrKeyScan_RegisterKeyEventHandler** (AMBA_IR_KEY_EVENT_HANDLER_f Handler)

Function Description:

- This function is used to add an IR Remote key event handler. When a system detects an IR button-press, all registered event handlers will be called.

Parameters:

Type	Parameter	Description
AMBA_IR_KEY_EVENT_HANDLER_f	**Handler**	Pointer to Key Event Handler `typedef void (*AMBA_IR_KEY_EVENT_HANDLER_f)(UINT16 KeyCode);`

*Table 3-10. Parameters for ADC IR API **AmbaIrKeyScan_RegisterKeyEventHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 3-11. Returns for ADC IR API **AmbaIrKeyScan_RegisterKeyEventHandler()**.*

Example:

```
AmbaIrKeyScan_RegisterKeyEventHandler(IrKeyScanHandler);
```

See Also:

AmbaIrKeyScan_UnRegisterKeyEventHandler()

3.3.6 AmbaIrKeyScan_UnRegisterKeyEventHandler

API Syntax:

int **AmbaIrKeyScan_UnRegisterKeyEventHandler** (AMBA_IR_KEY_EVENT_HANDLER_f Handler)

Function Description:

- This function is used to remove an IR key event handler.

Parameters:

Type	Parameter	Description
AMBA_IR_KEY_EVENT_HANDLER_f	**Handler**	Pointer to Key Event Handler `typedef void (*AMBA_IR_KEY_EVENT_HANDLER_f)(UINT16 KeyCode);`

*Table 3-12. Parameters for ADC IR API **AmbaIrKeyScan_UnRegisterKeyEventHandler()**.*

Returns:

Return	Description
0	Success
- 1	Failure

*Table 3-13. Returns for ADC IR API **AmbaIrKeyScan_UnRegisterKeyEventHandler()**.*

Example:

```
AmbaIrKeyScan_UnRegisterKeyEventHandler(IrKeyScanHandler);
```

See Also:

AmbaIrKeyScan_RegisterKeyEventHandler()

Appendix 1 Additional Resources

Related resources include:

- *SDK6 API System*
- *SDK6 API AmbaKAL*

Please contact an Ambarella representative for a full list of related resources.

Appendix 2 Important Notice

All Ambarella design specifications, datasheets, drawings, files, and other documents (together and separately, "materials") are provided on an "*as is*" basis, and Ambarella makes no warranties, expressed, implied, statutory, or otherwise with respect to the materials, and expressly disclaims all implied warranties of noninfringement, merchantability, and fitness for a particular purpose. The information contained herein is believed to be accurate and reliable. However, Ambarella assumes no responsibility for the consequences of use of such information.

Ambarella Incorporated reserves the right to correct, modify, enhance, improve, and otherwise change its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

All products are sold subject to Ambarella's terms and conditions of sale supplied at the time of order acknowledgment. Ambarella warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with its standard warranty. Testing and other quality control techniques are used to the extent Ambarella deems necessary to support this warranty.

Ambarella assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using Ambarella components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

Ambarella does not warrant or represent that any license, either expressed or implied, is granted under any Ambarella patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Ambarella products or services are used. Information published by Ambarella regarding third-party products or services does not constitute a license from Ambarella to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from Ambarella under the patents or other intellectual property of Ambarella.

Reproduction of information from Ambarella documents is not permissible without prior approval from Ambarella.

Ambarella products are not authorized for use in safety-critical applications (such as life support) where a failure of the product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Customers acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of Ambarella products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by Ambarella. Further, Customers must fully indemnify Ambarella and its representatives against any damages arising out of the use of Ambarella products in such safety-critical applications.

Ambarella products are neither designed nor intended for use in military/aerospace applications or environments. Customers acknowledge and agree that any such use of Ambarella products is solely at the Customer's risk, and they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

Appendix 3 Revision History

NOTE: Page numbers for previous drafts may differ from page numbers in the current version.

Version	Date	Comments
0.1	8 April 2013	Initial draft
1.0	19 April 2013	Preliminary release
1.1	25 October 2013	Refine descriptions; update formatting
1.2	28 October 2014	Update function AmbaIrKeyScan_Init in section 3.3.4
1.3	17 September 2014	Formatted to SDK6

Table A3-1. Revision History.