

Overview

Cryptography enables the confidentiality of communication through an insecure channel and hence protects against unauthorized parties by preventing unauthorized alteration of use. The Advanced Encryption Standard, AES(Rijndael) algorithm is implemented as described in the NIST (National Institute of Standards and Technology) Federal Information Processing Standard. This design example implements the 128-bit block-size AES algorithm, which accepts a 128-bit plain data input word, and generates a corresponding 128-bit cipher output word using a supplied 128, 192, or 256-bit AES key.

Features

- Encrypts using the AES Rijndael Block Cipher Algorithm
- Processes 128-bit data blocks with 32-bit data interface
- Parameterizable key length(can be 128, 192 or 256)
- Includes the key expansion function and s-box memory
- Completely self-contained: does not require external memory
- Internal Buffer to hold encrypted data
- IP-XACT version 1.2 compliant

Features not supported

- 128-bit data path for high speed systems
- Support for all Cipher Modes - CBC,CFB,OFB,CTR
- Generic 8-bit, 16-bit, 64-bit, 128-bit external interface

Resource Utilization

Table 1: Resource Utilization

LUTs	Registers	Memory	I/Os	GBs
2616	1973	0	0	0

Note: Resource Utilization is based on iCEcube 2010.12.14671 release.

System Block Diagram

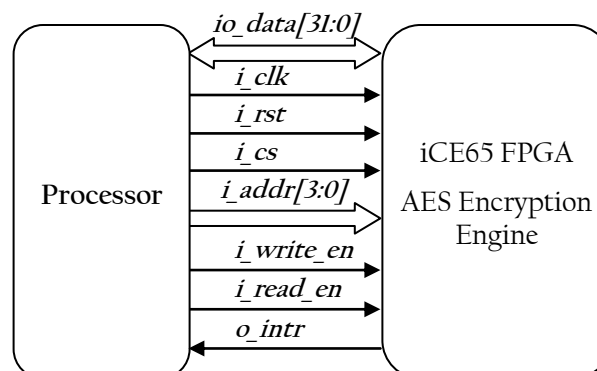


Figure 1: System Block Diagram

Functional Block Diagram

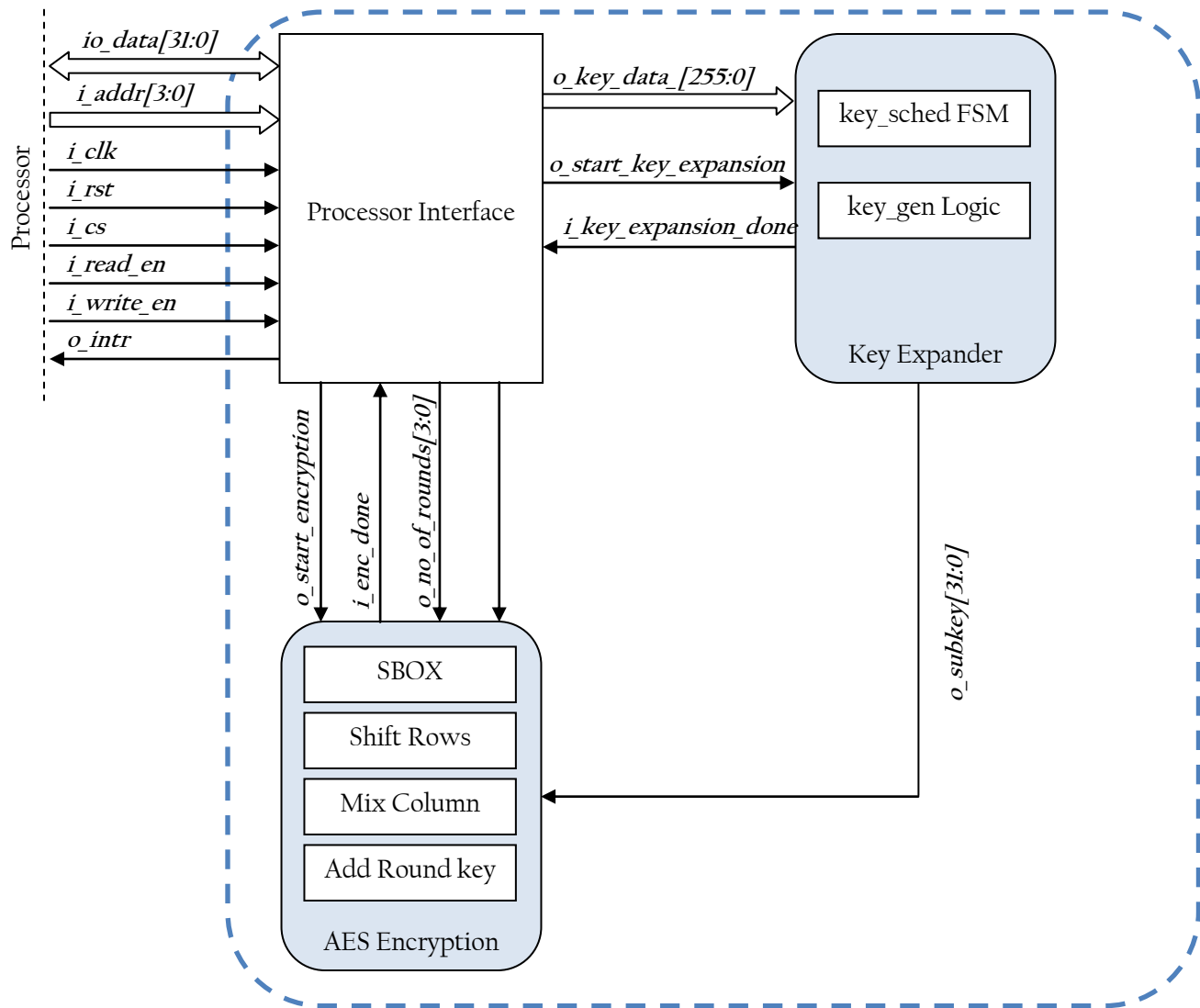


Figure 2: Functional Block Diagram

Design Interface

Table 2: Pin Description

Signal Name	Pin Type	Signal Description
i_clk	Input	System clock at 32MHz
i_rst	Output	Active high asynchronous system reset
i_cs	Input	Active high chip select from processor
i_read_en	Output	Active high read enable from processor
i_write_en	Input	Active high write enable from processor
o_intr	Output	Active high interrupt to processor
i_addr[3:0]	Input	Register Address.
io_data[31:0]	Output	Data bus.

Configurable Parameters

- **key_length** – This parameter configures the length of the key to be used. Its default value is 256. It can be configured to 192 or 128 as well.

Register Map

There are a set of predefined registers, which controls the data flow and operation of AES Engine. Table 3 summarizes the available registers and gives a brief description.

Table 3: AES Register and Address Mapping

Address	Register	Description
0X0	AES_STATUS	AES Engine Status Register
0x4	AES_CONTROL	AES Engine Control Register
0x8	AES_DATA	Write and Read Register to/from the AES Engine
0XC	AES_KEY	Write Register – key data

AES Status Register

This read-only register holds the basic status information of the AES Engine. The Table 4 shows the contents of the AES status register in detail. All the status bits are active high ('1').

Table 4: AES Status Register

BIT31-6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
X	CRYPTDONE	EXPDONE	IDATFULL	KEYFULL	RST	BUSY

- **BUSY**: This bit is set during encoding, decoding and during key expansion.
- **RST**: This bit is set after a reset through the AES_CONTROL register and will be deactivated with the first valid write access to one of the AES registers.
- **KEYFULL**: This bit is set as soon as processor writes 128/192/256 bits of key to the Key data register. Power-On reset or RESET command clears KEYFULL bit of status register. Also, this status bit is cleared as soon as processor starts writing to AES Key register and goes active as soon as AES Engine receives the necessary number of bits defined in KEY LENGTH of Control Register.
- **IDATFULL**: This bit is set as soon as processor writes 128 bits of Data to the Data register. Power-On reset or RESET command clears IDATFULL bit of status register. Also, this status bit is cleared as soon as processor starts writing to AES Data register and goes active as soon as AES Engine receives 128-bits of data. This bit again cleared when AES Engine read whole 128-bits of data and started processing it from buffer. A new set of data could be written to data register when it is cleared.
- **EXPDONE**: Indicates completion of Key Expansion process. Processor can check this bit when AES Engine generates an interrupt.
- **CRYPTDONE**: Indicates completion of encryption process. Processor can check this bit when AES engine generates an interrupt. This bit cleared as soon as processor starts reading from AES data register.

AES Control Register

This read-write register controls the operation of AES Engine. The Table 5 shows the contents of the AES control register in detail.

Table 5: AES Control Register

BIT31-2	BIT1	BIT0
X	COMMAND WORD	

A command word defines the required action of the AES Engine. RESET operation clears KEY, Data, Status and Control Registers. Valid command words are listed in Table 6 Control Register – Command Field.

Table 6: Control Register – Command Field

COMMAND	DESCRIPTION
00	NOP
01	RESET
10	KEY SETUP
11	AES ENCRYPT

Note that (1). Requesting an action during active operation (busy) of the AES Engine may corrupt the results of this operation. (2). changing the key length during active operation (busy) of the AES core may corrupt the results of this operation. There is no need to reset the contents prior sending a new command.

AES Key Register

The AES key write-only register is used to write key data to the AES Engine, MSB word first mode.

AES Data Register

The AES Data read-write register is used to write data to and read data from the AES Engine. 128 bit cipher and plain data is transferred to and from the AES Engine by sending/reading the MSB word first.

Design Details

AES Encryption is the process of transforming information using Rijndael block cipher algorithm to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The Encrypted data can only be deciphered if one has the password or the Key. The Encryption algorithm is a set of well defined steps to transform data from a readable format to an encoded format using the Key. The result of the process is encrypted information, referred to as cipher text.

AES is an iterated block cipher with a fixed block size of 128 and a variable key length. The AES algorithm operates on 128 bits of data and generates 128 bits of output. The length of the key used to encrypt this input data can be 128, 192 or 256 bits. AES has been designed as a substitution-permutation network (SPN) and encryption process uses 10, 12 and 14 encryption rounds for key length of 128, 192 and 256 bits respectively.

The AES Engine consists of three units. The Processor Interface unit holds all the registers required for the communication with processor. The Key Expander module holds and manipulates the key data. The AES Encryption module holds data as well as AES state and performs the AES transformations like ciphering.

Processor Interface

An Interface between the processor and the AES system is based on a generic parallel bus protocol. This processor interface controls the movement of control commands, address and data between the processor and the AES Engine. Since the data movement uses 32-bit mode, it has to be de-serialized into a 128-bit data and sent to Encryption engine. In a similar fashion, 128-bit data from the AES Engine is serialized (32-bit mode) before sending it to the processor.

This module is also responsible for

- Generating control signals to trigger key expansion and encryption process.
- Supply the data and key from the its buffer to the appropriate modules depending upon the control signals.
- Write the encrypted data to its internal buffer from appropriate modules depending upon the control signals.
- Generate an interrupt to the processor when AES Engine completes Key Expansion and AES Encryption of 128-bits of data.

Key Expansion:

Key Expansion operation is triggered when control register receives KEY_SETUP command word. The AES algorithm takes the Cipher Key K, and performs a Key Expansion routine to generate a key schedule. Nb defines the number of columns of 32 bits in the 128-bit data, which is $Nb = 128/32 = 4$. Similarly, Nk defines the number of columns of 32 bits of key, which is, $Nk = 128/32 = 4$. For key length of 192 and 256 the values of Nk will be $192/32 = 6$ and $256/32 = 8$ respectively. The number of rounds Nr =10 when Nk= 4 and changes to 12 and 14 for Nk =6 and Nk =8. The Key Expansion generates a total of Nb (Nr + 1) words: the algorithm requires an initial set of Nk words, and each of the Nr rounds requires Nk words of key data. As soon as key expansion process is finished, the BUSY bit and EXPDONE bits of the AES status register is set and processor is interrupted after which processor checks the status of the AES engine and determines the AES engine has finished key expansion process.

AES Encryption :

The AES Encryption process works as follows: 1). Processor writes the key data to AES Key register. Now the key setup operation has to be performed. The key setup operation is requested by writing the KEY_SETUP command to the AES Control register. The completion of the key setup operation is flagged by the interrupt signal and the busy bit in the AES Status register. 2). Processor writes plain data to AES Data register. The 128 bit input data block is delivered by four subsequent write commands to the AES Data register. Finally the encoding operation is started by writing the command AES_ENCRYPT to the AES Control register. The completion of the encoding operation is flagged by the interrupt signal and the busy bit in the AES status register. Now the encrypted data may be read from the AES Data register by four subsequent read commands. The key remains in the module until a new key is written or a reset command is performed.

Timing Diagrams

Signals definition:

i_clk – System Clock

i_write_en – Active high write enable

i_addr[3:0] – Register Address

io_data[31:0] – Data Bus

i_read_en – Active high read enable

KEY SETUP, EXPDONE, BUSY – contents of command register of AES Engine

AES ENCRYPT, CRYPTDONE – contents of command/status registers of AES Engine

The following figure illustrates a data write transaction.

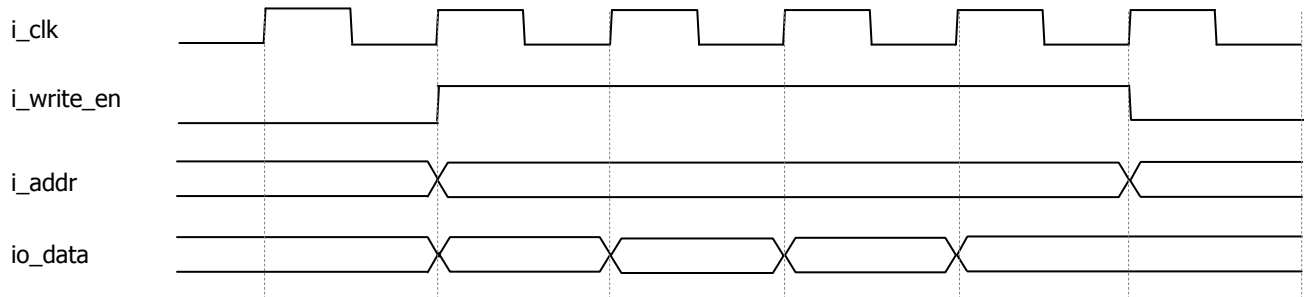


Figure 3: AES Data write access

The following figure illustrates a data read transaction.

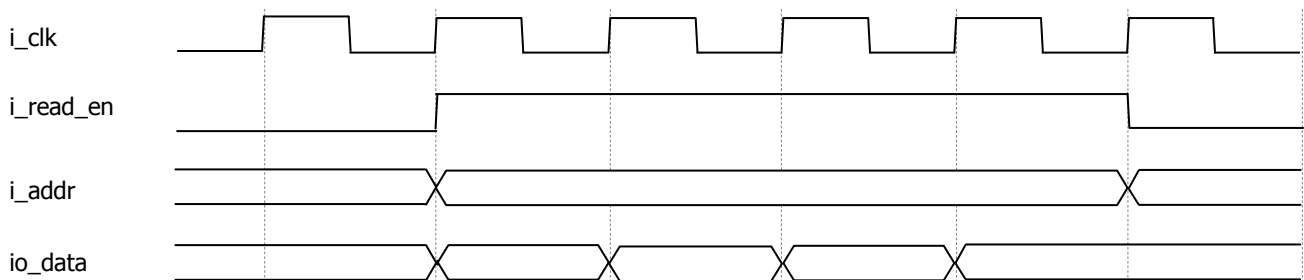


Figure 4: AES Data Read access

The following figure illustrates the key expansion process.

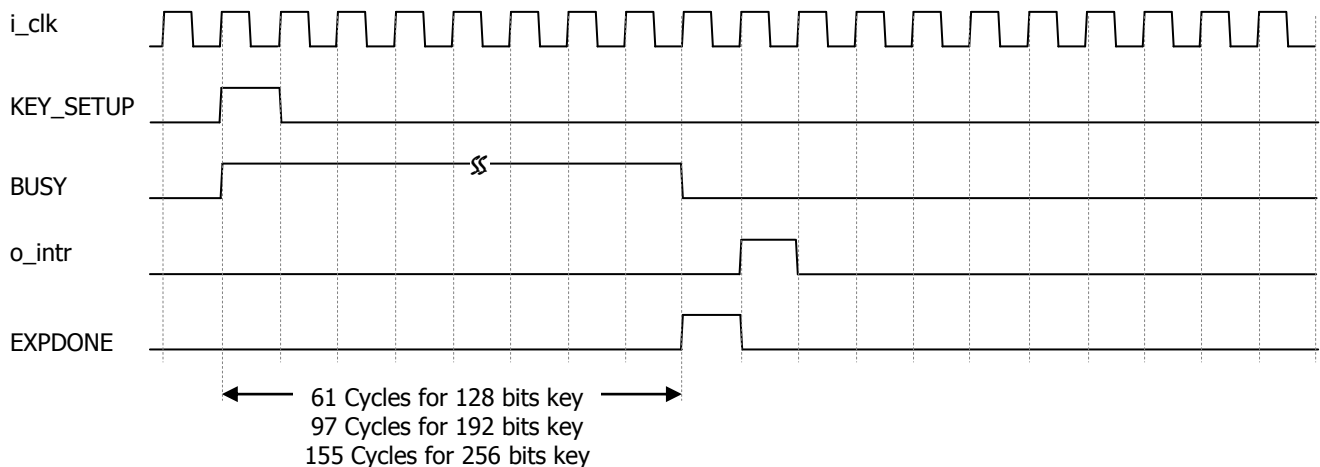


Figure 5: Timing Diagram of Key Expansion Process

The following figure illustrates the AES Encryption process

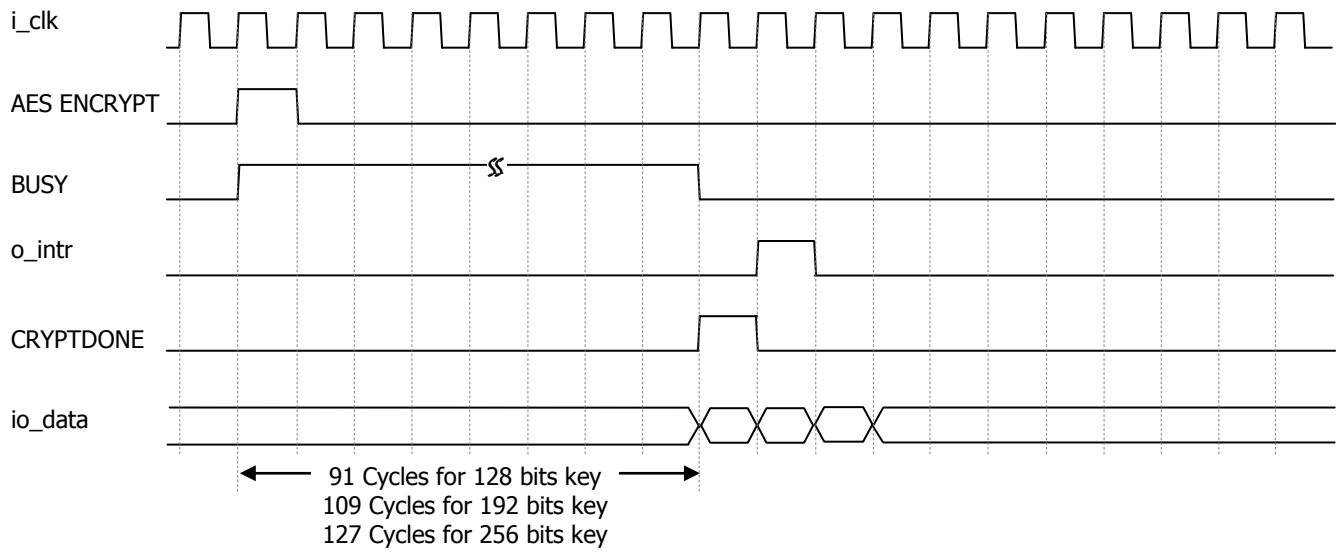
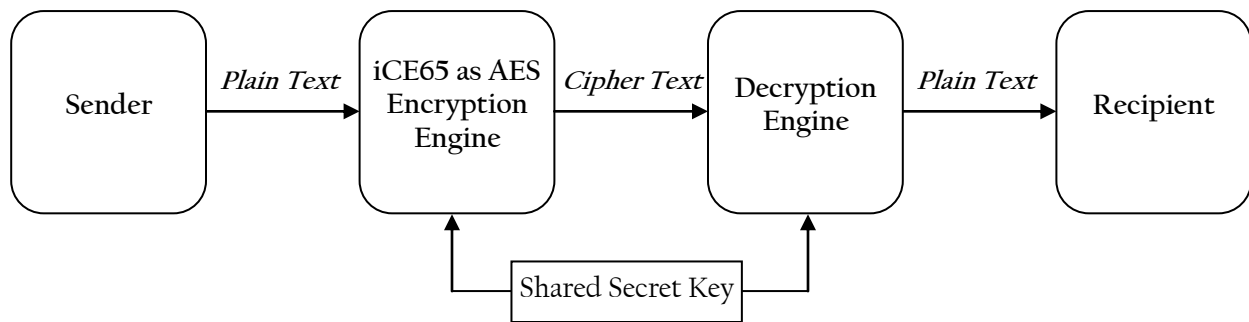


Figure 6: Timing Diagram of AES Encryption Process

Usage Examples

Example #1



The sender creates a cipher text message by encrypting the plain text message with the iCE65 AES Encryption Engine and a shared secret key. The sender writes the key data to AES Key register. Then the key setup operation is performed. Then the sender writes plain data. The 128 bit input data block is delivered by four subsequent write commands to the AES Data register. This is followed by the encoding operation. Now the encrypted data may be read from the AES Data register by four subsequent read commands.

The sender sends the cipher text message to the recipient.

The recipient decrypts the cipher text message back into plain text with the shared secret key.

System Designer Flow

AES Encryption is compatible with System Designer/IP-XACT 1.2. Following parameter can be configured in the System Designer environment.

- **key_length** – This parameter configures the length of the key to be used. Its default value is 256. It can be configured to 192 or 128 as well.

The System Designer flow is as follows,

1. Launch the System Designer from Synplify Pro using menu 'Import -> Launch System Designer'.
2. Create a new project (open an existing old project, as necessary) and import the IP-XACT XML file
3. Drag and place the component from the 'Library' pane to the 'Design' pane
4. To change the key length, right-click on the component instance, and click on "Open Configuration". Go to "Edit Instance Parameters" tab, change the "key_length" parameter. Click on the "Apply" button, and then close it.
5. Click on the "Generate Files" button, which generates the necessary files required for synthesis and simulation.
6. Go to Synplify Pro and click on the "Run" button to synthesize the System Designer generated files. Synplify Pro generates all the necessary files for P&R in iCECube.

References

The following references were used in the creation of this design:

- SiliconBlue Technologies, Inc. “[iCE65 Ultra Low-Power mobileFPGA Family](#)” datasheet (26-MAY-2010).
- National Institute of Standards and Technology, “[Federal Information Processing Standard 197, The Advanced Encryption Standard \(AES\)](#),” csrc.nist.gov/publications/fips/fips197/fips-197.pdf , 2001.

Revision History

Version	Date	Description
1.0	09-SEP-2010	Initial Draft Document
1.1	07-DEC-2010	IP-XACT format Update

Disclaimer

Copyright © 2007–2009 by SiliconBlue Technologies LTD. All rights reserved. SiliconBlue is a registered trademark of SiliconBlue Technologies LTD in the United States. Specific device designations, and all other words and logos that are identified as trademarks are, unless noted otherwise, the trademarks of SiliconBlue Technologies LTD. All other product or service names are the property of their respective holders. SiliconBlue products are protected under numerous United States and foreign patents and pending applications, maskwork rights, and copyrights. SiliconBlue warrants performance of its semiconductor products to current specifications in accordance with SiliconBlue's standard warranty, but reserves the right to make changes to any products and services at any time without notice. SiliconBlue assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by SiliconBlue Technologies LTD. SiliconBlue customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



SiliconBlue Technologies Corporation

3255 Scott Blvd.,
Building 7, Suite 101
Santa Clara, CA 95054

Tel: 408-727-6101
Fax: 408-727-6085

www.SiliconBlueTech.com