Soft ECC Proxy v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG337 (v1.0) August 13, 2021

Table of Contents

Introduction

As technology is evolving, the dynamic random-access memory (DRAM) device size is increasing and the components on the chips are getting smaller. Because of this there is an increase in the electrical or magnetic interference on the DRAM chips. Lower energy particles are able to change the memory cell's state. These kinds of interferences can cause a single bit of DRAM to spontaneously flip to the opposite state. This can either lead the system to crash or to corrupt data. In addition, applications addressing the functional safety systems require the mitigation of data and address. This is done not only for the failures induced by interferences, but also for those induced by permanent faults like stuck-at and shorts.

Several approaches have been developed to deal with these unwanted bit-flips. One of them is to calculate an error-correction code (ECC) for the data and store it in the DRAM along with the data. The most common ECC, a SECDED Hamming code, allows you to correct single-bit errors and detect double bit errors.

Features

- Supports AXI4 and AXI4-Lite.
- Configurable data width from 8 to 512. The data width must be power of 2.
- ECC calculation for every data byte.
- Single bit error detection and correction, double bit error detection for every data byte.
- Supports both soft and hard reset.
- Configurable outstanding transaction support up to 64-bits for read channel.
- When ECC is applied, the value of the data width doubles.
- In case of a single-bit error, the bit is corrected along with the correction being notified.
- Supports Two Types of ECC modes: Hamming and HSIAO.
- Error Injection(Single and Double bit error).
- Slave error response incase of Error detection.
- Error Start address capturing in registers.
- Provides error count for both single and double bit errors.

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Versal™ ACAP
	UltraScale+™ Families
	UltraScale™ Architecture
	Zynq®-7000
	7 series
Supported User Interfaces	AXI4, AXI4-Lite
Resources	Performance and Resource Use web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	Not Provided
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado® IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

XILINX

Chapter 2

Overview

Navigating Content by Design Process

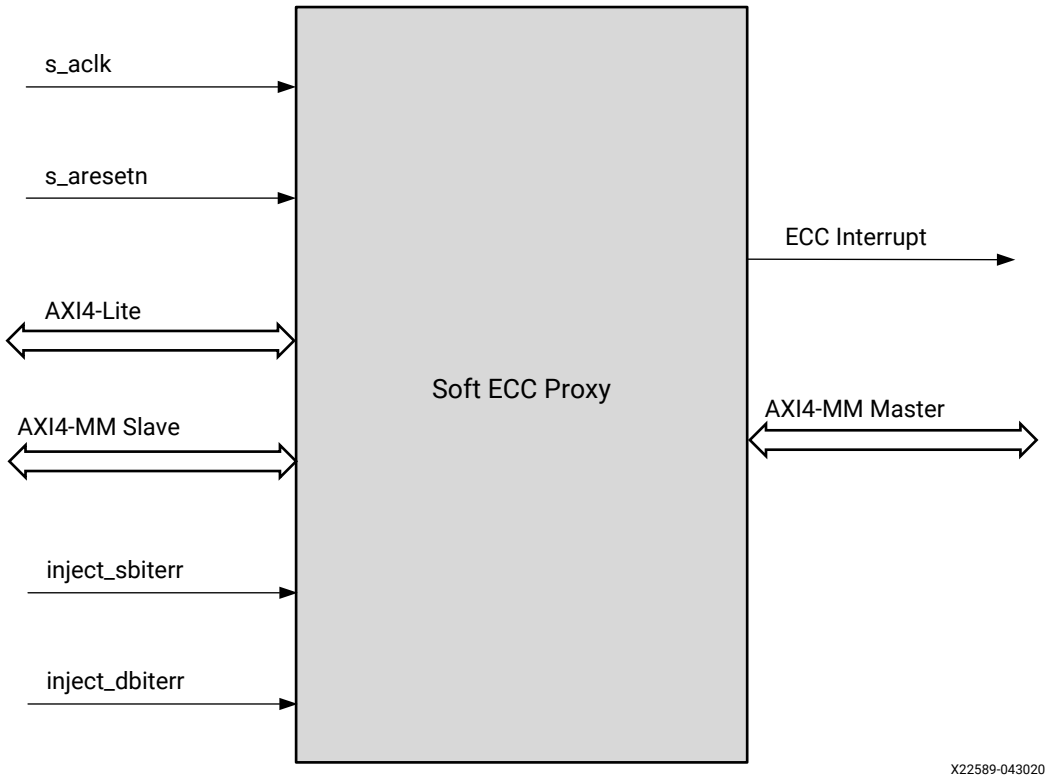
Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal™ ACAP design process Design Hubs can be found on the Xilinx.com website. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Port Descriptions
  - Register Space
  - Clocking and Resets
  - Customizing and Generating the Core
  - Chapter 6: Example Design

Core Overview

This product guide describes features of the Xilinx® Soft ECC Proxy LogiCORE IP and the functionality of the various registers in the design. In addition, the core interface and its customization options are defined in the following sections.

Figure 1: **Soft ECC Proxy IP Basic Block Diagram**



Soft ECC Proxy IP consists of AXI4 memory mapped master (AXI4-MM Master) and slave (AXI4-MM Slave) interfaces. The ECC bits are calculated for every byte of write data from the slave. The calculated ECC bits puts on to master interface along with its data, which then reads from the master interface. The ECC decoding is done on the read data from master interface and generate interrupts (single/double bit error) for every byte of read data. The status of the interrupts can be accessed through the Interrupt Status Register. The Common ECC interrupt pin at the output represents the ECC error occurred. The register space inside the IP is accessed through the AXI4-Lite interface. In case of an interrupt, the type of error (single-bit or double-bit) can be read from the Interrupt Type Register. And the corresponding byte or bit is set in the Interrupt Status Register.

Applications

The Soft ECC Proxy core is used in applications such as end-to-end data protection.

Unsupported Features

The following features of the standard are not supported in the core:

- Address protection.

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

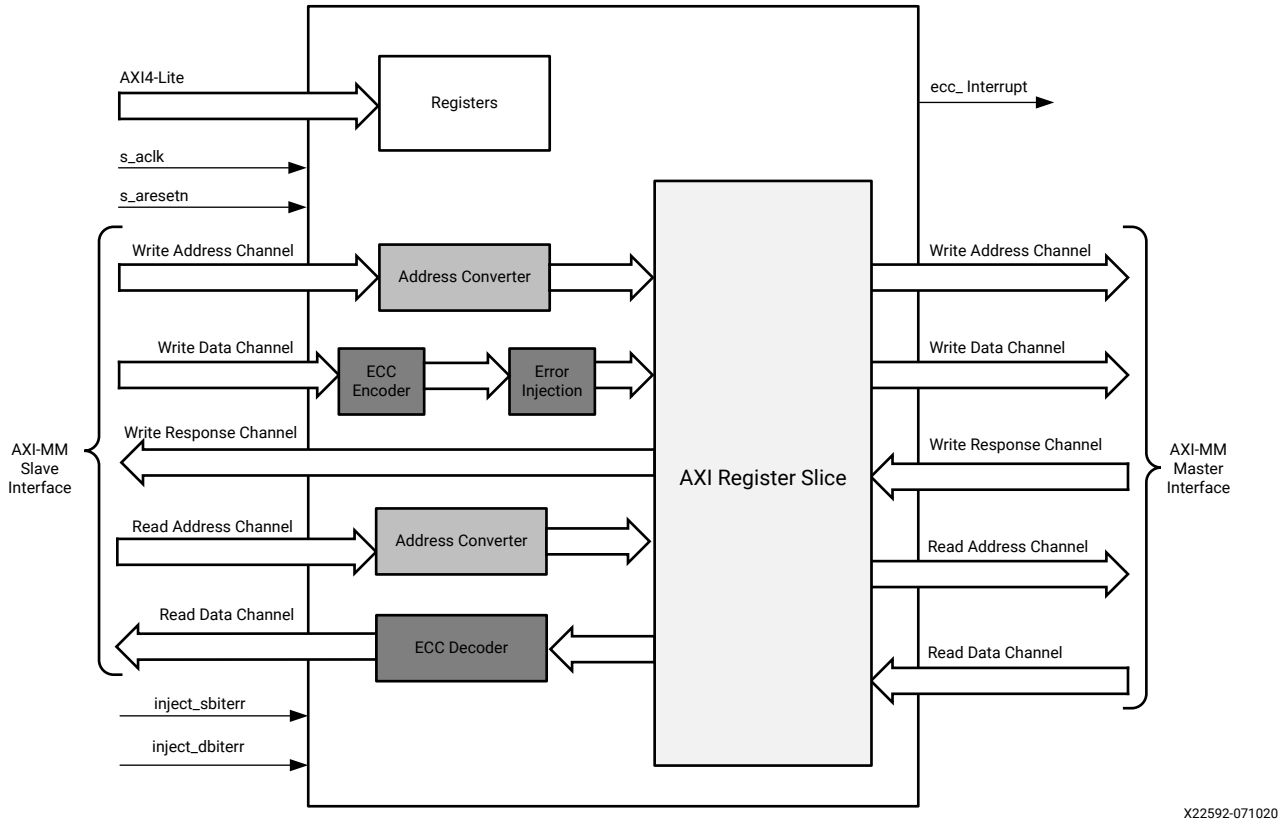
For more information about this core, visit the Soft ECC Proxy product web page.

Information about other Xilinx® LogiCORE™ IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

The functional block diagram of the core is shown in the following figure.

Figure 3: **Core Block Diagram**



Performance and Resource Use

For full details about performance and resource use, visit the Performance and Resource Use web page.

Port Descriptions

The core interfaces are explained in detail in the following sections.

Interface Ports

Table 1: **Interface Ports**

Port Name	I/O	Width	Description
s_aclk	Input	1	**Global Slave Interface Clock:** All signals are sampled on the rising edge of this clock.
s_aresetn	Input	1	Active-Low Asynchronous Reset
AXI4-Lite Interface			
s_axil_awaddr	Input	32	AXI4-Lite Write Address
s_axil_awprot	Input	3	AXI4-Lite Write Protection Type
s_axil_awvalid	Input	1	AXI4-Lite Write Address Valid
s_axil_awready	Output	1	AXI4-Lite Write Address Ready
s_axil_wdata	Input	32	AXI4-Lite Write Data
s_axil_wstrb	Input	4	AXI4-Lite Write Strobe
s_axil_wvalid	Input	1	AXI4-Lite Write Data Valid
s_axil_wready	Output	1	AXI4-Lite Write Data Ready
s_axil_bresp	Output	2	AXI4-Lite Write Response
s_axil_bvalid	Output	1	AXI4-Lite Write Response Valid
s_axil_bready	Input	1	AXI4-Lite Write Response Ready
s_axil_araddr	Input	32	AXI4-Lite Read Address
s_axil_arprot	Input	3	AXI4-Lite Read Protection Type
s_axil_arvalid	Input	1	AXI4-Lite Read Address Valid
s_axil_arready	Output	1	AXI4-Lite Read Address Ready
s_axil_rdata	Output	32	AXI4-Lite Read Data
s_axil_rresp	Output	2	AXI4-Lite Read Response
s_axil_rvalid	Output	1	AXI4-Lite Read Data Valid
s_axil_rready	Input	1	AXI4-Lite Read Data Ready
AXI4-Lite - MM Slave Interface (Write Address Channel)			
s_axi_awid	Input	C_ID_WIDTH	**Write Address ID:** Identification tag for the write address group of signals.
s_axi_awaddr	Input	C_ADDR_WIDTH	**Write Address:** The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.

Table 1: **Interface Ports** *(cont'd)*

Port Name	I/O	Width	Description
s_axi_awlen	Input	8	**Burst Length:** The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
s_axi_awsize	Input	3	**Burst Size:** Indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.
s_axi_awburst	Input	2	**Burst Type:** The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
s_axi_awlock	Input	2	**Lock Type:** This signal provides additional information about the atomic characteristics of the transfer.
s_axi_awcache	Input	4	**Cache Type:** Indicates the buffer-able, cache-able, write-through, write-back, and allocate attributes of the transaction.
s_axi_awprot	Input	3	**Protection Type:** Indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
s_axi_awqos	Input	4	**Quality of Service (QoS):** Sent on the write address channel for each write transaction.
s_axi_awregion	Input	4	**Region Identifier:** Sent on the write address channel for each write transaction.
s_axi_awuser	Input	C_AWUSER_WIDTH	Write Address Channel User
s_axi_awvalid	Input	1	**Write Address Valid:** Indicates that valid write address and control information are available.
s_axi_awready	Output	1	**Write Address Ready:** Indicates that the slave is ready to accept an address and associated control signals.
AXI4-Lite - MM Slave Interface (Write Data Channel)			
s_axi_wdata	Input	C_DATA_WIDTH	**Write Data:** The write data bus can be 8, 16, 32, 64, 128, 256, or 512 bits wide.
s_axi_wstrb	Input	C_DATA_WIDTH/8	**Write Strobes:** Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus.
s_axi_wlast	Input	1	**Write Last:** Indicates the last transfer in a write burst.
s_axi_wuser	Input	C_WUSER_WIDTH	Write Data Channel User

Table 1: **Interface Ports** *(cont'd)*

Port Name	I/O	Width	Description
s_axi_wvalid	Input	1	**Write Valid:** Indicates that valid write data and strobes are available.
s_axi_wready	Output	1	**Write Ready:** Indicates that the slave can accept the write data.
AXI4-Lite - MM Slave Interface (Write Response Channel)			
s_axi_bid	Output	C_ID_WIDTH	**Response ID:** The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
s_axi_bresp	Output	2	**Write Response:** Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
s_axi_buser	Output	C_BUSER_WIDTH	Write Response Channel User
s_axi_bvalid	Output	1	**Write Response Valid:** Indicates that a valid write response is available.
s_axi_bready	Input	1	**Response Ready:** Indicates that the master can accept the response information.
AXI4-Lite - MM Slave Interface (Read Address Channel)			
s_axi_arid	Input	C_ID_WIDTH	**Read Address ID:** This signal is the identification tag for the read address group of signals.
s_axi_araddr	Input	C_ADDR_WIDTH	**Read Address:** The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
s_axi_arlen	Input	8	**Burst Length:** The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
s_axi_arsize	Input	3	**Burst Size:** This signal indicates the size of each transfer in the burst.
s_axi_arburst	Input	2	**Burst Type:** The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
s_axi_arlock	Input	2	**Lock Type:** This signal provides additional information about the atomic characteristics of the transfer.

Table 1: **Interface Ports** *(cont'd)*

Port Name	I/O	Width	Description
s_axi_arcache	Input	4	**Cache Type:** This signal provides additional information about the cache-able characteristics of the transfer.
s_axi_arprot	Input	3	**Protection Type:** This signal provides protection unit information for the transaction.
s_axi_arqos	Input	4	**Quality of Service (QoS):** Sent on the read address channel for each read transaction.
s_axi_arregion	Input	4	**Region Identifier:** Sent on the read address channel for each read transaction.
s_axi_aruser	Input	C_ARUSER_WIDTH	Read Address Channel User
s_axi_arvalid	Input	1	**Read Address Valid:** When High, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, `aready` is high.
s_axi_arready	Output	1	**Read Address Ready:** Indicates that the slave is ready to accept an address and associated control signals.
AXI4-Lite - MM Slave Interface (Read Data Channel)			
s_axi_rid	Output	C_ID_WIDTH	**Read ID Tag:** ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
s_axi_rdata	Output	C_DATA_WIDTH	**Read Data:** Can be 8, 16, 32, 64, 128, 256, or 512 bits wide.
s_axi_rresp	Output	2	**Read Response:** Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
s_axi_rlast	Output	1	**Read Last:** Indicates the last transfer in a read burst.
s_axi_ruser	Output	C_RUSER_WIDTH	Read Data Channel User
s_axi_rvalid	Output	1	**Read Valid:** Indicates that the required read data is available and the read transfer can complete.
s_axi_rready	Input	1	**Read Ready:** Indicates that the master can accept the read data and response information.
AXI4-Lite - MM Master Interface (Write Address Channel)			
m_axi_awid	Output	C_ID_WIDTH	**Write Address ID:** Identification tag for the write address group of signals.

Table 1: **Interface Ports** *(cont'd)*

Port Name	I/O	Width	Description
m_axi_awaddr	Output	C_ADDR_WIDTH+1	**Write Address:** The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
m_axi_awlen	Output	8	**Burst Length:** The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
m_axi_awsize	Output	3	**Burst Size:** Indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.
m_axi_awburst	Output	2	**Burst Type:** The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
m_axi_awlock	Output	2	**Lock Type:** This signal provides additional information about the atomic characteristics of the transfer.
m_axi_awcache	Output	4	**Cache Type:** Indicates the buffer-able, cache-able, write-through, write-back, and allocate attributes of the transaction.
m_axi_awprot	Output	3	**Protection Type:** Indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
m_axi_awqos	Output	4	**Quality of Service (QoS):** Sent on the write address channel for each write transaction.
m_axi_awregion	Output	4	**Region Identifier:** Sent on the write address channel for each write transaction.
m_axi_awuser	Output	C_AWUSER_WIDTH	Write Address Channel User
m_axi_awvalid	Output	1	**Write Address Valid:** Indicates that valid write address and control information are available.
m_axi_awready	Input	1	**Write Address Ready:** Indicates that the slave is ready to accept an address and associated control signals.
AXI4-Lite - MM Slave Interface (Write Data Channel)			
m_axi_wdata	Output	2*C_DATA_WIDTH	**Write Data:** The write data bus can be 8, 16, 32, 64, 128, 256, or 512 bits wide.

Table 1: **Interface Ports** *(cont'd)*

Port Name	I/O	Width	Description
m_axi_wstrb	Output	2*C_DATA_WIDTH/8	**Write Strobes:** Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus.
m_axi_wlast	Output	1	**Write Last:** Indicates the last transfer in a write burst.
m_axi_wuser	Output	C_WUSER_WIDTH	Write Data Channel User
m_axi_wvalid	Output	1	**Write Valid:** Indicates that valid write data and strobes are available.
m_axi_wready	Input	1	**Write Ready:** Indicates that the slave can accept the write data.
AXI4-Lite - MM Slave Interface (Write Response Channel)			
m_axi_bid	Input	C_ID_WIDTH	**Response ID:** The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
m_axi_bresp	Input	2	Write Response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
m_axi_buser	Input	C_BUSER_WIDTH	Write Response Channel User
m_axi_bvalid	Input	1	**Write Response Valid:** Indicates that a valid write response is available.
m_axi_bready	Output	1	**Response Ready:** Indicates that the master can accept the response information.
AXI4-Lite - MM Slave Interface (Read Address Channel)			
m_axi_arid	Output	C_ID_WIDTH	**Read Address ID:** This signal is the identification tag for the read address group of signals.
m_axi_araddr	Output	C_ADDR_WIDTH+1	**Read Address:** The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
m_axi_arlen	Output	8	**Burst Length:** The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
m_axi_arsize	Output	3	**Burst Size:** This signal indicates the size of each transfer in the burst.

Table 1: **Interface Ports** *(cont'd)*

Port Name	I/O	Width	Description
m_axi_arburst	Output	2	**Burst Type:** The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
m_axi_arlock	Output	2	**Lock Type:** This signal provides additional information about the atomic characteristics of the transfer.
m_axi_arcache	Output	4	**Cache Type:** This signal provides additional information about the cache-able characteristics of the transfer.
m_axi_arprot	Output	3	**Protection Type:** This signal provides protection unit information for the transaction.
m_axi_arqos	Output	4	**Quality of Service (QoS):** Sent on the read address channel for each read transaction.
m_axi_arregion	Output	4	**Region Identifier:** Sent on the read address channel for each read transaction.
m_axi_aruser	Output	C_ARUSER_WIDTH	Read Address Channel User
m_axi_arvalid	Output	1	**Read Address Valid:** When High, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, `aready` is high.
m_axi_arready	Input	1	**Read Address Ready:** Indicates that the slave is ready to accept an address and associated control signals.
AXI4-Lite - MM Slave Interface (Read Data Channel)			
m_axi_rid	Input	C_ID_WIDTH	**Read ID Tag:** ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
m_axi_rdata	Input	2*C_DATA_WIDTH	**Read Data:** Can be 8, 16, 32, 64, 128, 256, or 512 bits wide.
m_axi_rresp	Input	2	**Read Response:** Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
m_axi_rlast	Input	1	**Read Last:** Indicates the last transfer in a read burst.
m_axi_ruser	Input	C_RUSER_WIDTH	Read Data Channel User
m_axi_rvalid	Input	1	**Read Valid:** Indicates that the required read data is available and the read transfer can complete.

Table 1: **Interface Ports** *(cont'd)*

Port Name	I/O	Width	Description
m_axi_rready	Output	1	**Read Ready:** Indicates that the master can accept the read data and response information.
Error Inject Signals			
inject_sbiterr	Input	1	Inject Single-Bit Error
inject_dbiterr	Input	1	Inject Double-Bit Error
Interrupts			
ecc_interrupt	Output	1	**ECC Interrupt:** Indicates single/ double bit error occurred on read data path

Register Space

The Soft ECC Proxy IP provides registers to control its behavior and also provides its status information. The register space in this IP can be accessed through the AXI4-Lite interface. The following are the list of existing registers in this IP.

Table 2: **Register Address Space**

Offset (hex)	Register Name	Access Type	Reset Value	Description
0x00	Soft Reset Register	Write Only	0x0	Soft Reset Register
0x04	Interrupt type Register	Read Only	0x0	Interrupt Type Register: This register gives the information on type of error occurred (single-bit/double-bit).
0x10	Interrupt Status Register 0	Read Only	0x0	Gives the status of Single bit error occurred on data bytes 0 to 31.
0x14	Interrupt Status Register 1	Read Only	0x0	Gives the status of Single bit error occurred on data bytes 32 to 63.
0x20	Interrupt Status Register 2	Read Only	0x0	Gives the status of Double bit error occurred on data bytes 0 to 31.
0x24	Interrupt Status Register 3	Read Only	0x0	Gives the status of Double bit error occurred on data bytes 32 to 63.
0x28	Error Start Address Register	Read Only	0x0	Gives the lower 32 bits of the Busrt address when the first error occurs.
0x2C	Error Start Address Register 1	Read Only	0x0	Gives the higher 32 bits of the Busrt address when the first error occurs.
0x30	Single Bit Error Count Register	Read Only	0x0	Gives the number of single bit errors occured
0x34	Double Bit Error Count Register	Read Only	0x0	Gives the number of double bit errors occured

Soft Reset Register (0x00)

Soft reset register is to provide soft reset from the software.

Table 3: **Soft Reset Register (0x00)**

Bit	Name	Default Value	Access Type	Description
31-1	Reserved	NA	NA	Reserved
0	Soft Reset	0	Write Only	**Soft Reset**: When High, soft resets the IP

Interrupt Type Register (0x04)

This register provides information on the type of interrupt/error occurred.

Table 4: **Interrupt Type Register (0x04)**

Bit	Name	Default Value	Access Type	Description
31-2	Reserved	NA	NA	Reserved
1	Double Bit Error	0	Read Only	Occurrence of Double Bit Error: This bit is High if any of the data bytes have double bit error.
0	Single Bit Error	0	Read Only	Occurrence of Single Bit Error: This bit is High if any of the data bytes have single bit error.

Interrupt Status Register 0 (0x10)

Interrupt Status Register 0 gives the status of Single bit error occurred on data bytes 0 to 31.

Table 5: **Interrupt Status Register 0 (0x10)**

Bit	Name	Default Value	Access Type	Description
N	SBE_BYTE_N	0	Read Only	Single Bit Error on Byte N. Gives the status of Single bit error occurred on data bytes 0 to 31. For example:: Bit 0 corresponds to data byte 0. Bit 1 corresponds to data byte 1. If Bit 0 is high, means Single Bit error occurred on data byte0.

Interrupt Status Register 1 (0x14)

Interrupt Status Register 1 gives the status of single-bit error occurred on data bytes 32 to 63.

Table 6: **Interrupt Status Register 1 (0x14)**

Bit	Name	Default Value	Access Type	Description
N	SBE_BYTE_N	0	Read Only	Single Bit Error on Byte N. Gives the status of single-bit error occurred on data bytes 32 to 63. For example: Bit 0 corresponds to data byte 32. Bit 1 corresponds to data byte 33. If bit 0 is high, means single-bit error occurred on data byte 32.

Interrupt Status Register 2 (0x20)

Interrupt Status Register 2 gives the status of double-bit error occurred on data bytes 0 to 31.

Table 7: **Interrupt Status Register 2 (0x20)**

Bit	Name	Default Value	Access Type	Description
N	DBE_BYTE_N	0	Read Only	Double Bit Error on Byte N. Gives the status of double-bit error occurred on data bytes 0 to 31. For example: Bit 0 corresponds to data byte 0. Bit 1 corresponds to data byte 1. If Bit 0 is high, means Double Bit error occurred on data byte 0.

Interrupt Status Register 3 (0x24)

Interrupt Status Register 3 gives the status of Single bit error occurred on data bytes 32 to 63.

Table 8: **Interrupt Status Register 3 (0x24)**

Bit	Name	Default Value	Access Type	Description
N	DBE_BYTE_N	0	Read Only	Double Bit Error on Byte N. Gives the status of Double bit error occurred on data bytes 32 to 63. For example: Bit 0 corresponds to data byte 32. Bit 1 corresponds to data byte 33. If Bit 0 is high, means Double Bit error occurred on data byte32.

Error Start Address Register (0x28)

Error Start Address Register gives the lower 32-bits of burst address of the first error ocured.

Table 9: **Error Start Address Register (0x28)**

Bit	Name	Default Value	Access Type	Description
31:0	START_ADDRESS	0	Read Only	It captures the Start address (Burst Address) of the error occurred. It clears when read. This register read will also clears fields of the other register (Start Address Register1, Error Count registers).

Error Start Address Register 1 (0x2C)

Error Start Address Register gives the higher 32-bits of burst address of the first error ocured.

Table 10: **Error Start Address Register 1 (0x2C)**

Bit	Name	Default Value	Access Type	Description
31:0	START_ADDRESS	0	Read Only	It captures the Start address(Burst Address) of the error occurred. Reading this register will not clear the contents. This register cleared after reading Error Start Address Register (0x28).

Single Bit Error Count Register (0x30)

The Single Bit Error Count Register gives the number of beats that has the single bit errors starting from error start address.

Table 11: **Single Bit Error Count Register (0x30)**

Bit	Name	Default Value	Access Type	Description
31:0	ERROR_COUNT	0	Read Only	Number of beats that have the single bit errors starting from error start address. Reading this register will not clear the contents. This register cleared after reading Error Start Address Register (0x28).

Double Bit Error Count Register (0x34)

The Double Bit Error Count Register gives the number of beats that has the double bit errors starting from error start address.

Table 12: **Double Bit Error Count Register (0x34)**

Bit	Name	Default Value	Access Type	Description
31:0	ERROR_COUNT	0	Read Only	Number of beats that have the double bit errors starting from error start address. Reading this register will not clear the contents. This register cleared after reading Error Start Address Register (0x28).

Designing with the Core

This section includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

The basic blocks in Soft ECC Proxy LogiCORE™ IP are ECC Encoder, ECC Decoder, and AXI Register Slice.

Use the Example Design

Each instance of the Soft ECC Proxy core created by the Vivado design tool is delivered with an example design that can be implemented in a device and then simulated. This design can be used as a starting point for your own design or can be used to sanity-check your application in the event of difficulty. See the Example Design content for information about using and customizing the example designs for the core.

Registering Signals

To simplify timing and increase system performance in a programmable device design, keep all inputs and outputs registered between the user application and the core. This means that all inputs and outputs from the user application should come from, or connect to, a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx® tools to place and route the design.

Make Only Allowed Modifications

You should not modify the core. Any modifications can have adverse effects on system timing and protocol compliance. Supported user configurations of the core can only be made by selecting the options in the customization IP dialog box when the core is generated.

Clocking

The Soft ECC Proxy IP core operates on a single clock (`s_aclk`), and all input and output interface signals of the AXI4-Lite and AXI4 Master/Slave interfaces are synchronized with this clock.

Resets

The Soft ECC Proxy IP core uses a single asynchronous reset (`s_aresetn`).

Protocol Description

This section describes the operation of the Soft ECC Proxy LogiCORE™ IP core.

Figure 6: **Core Write Ports**

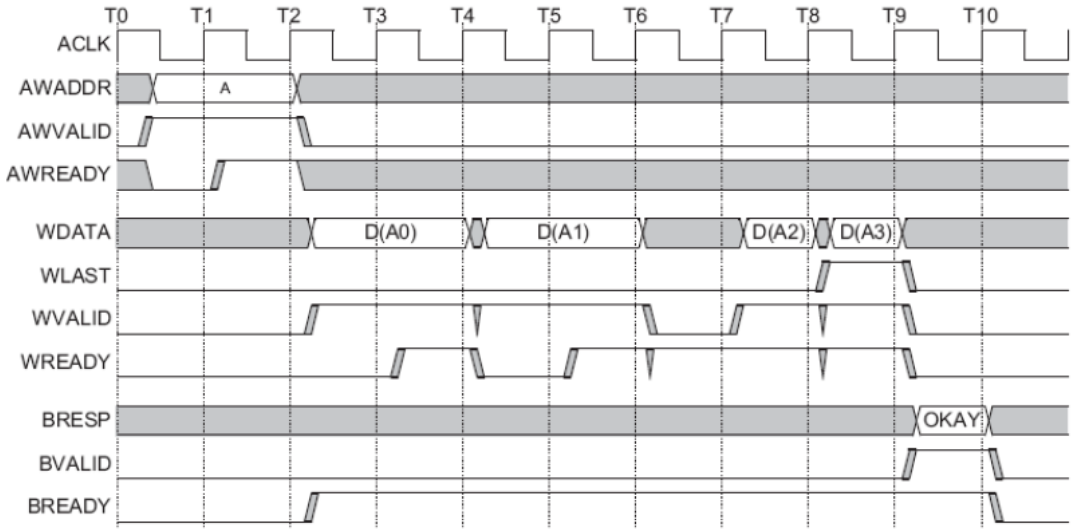
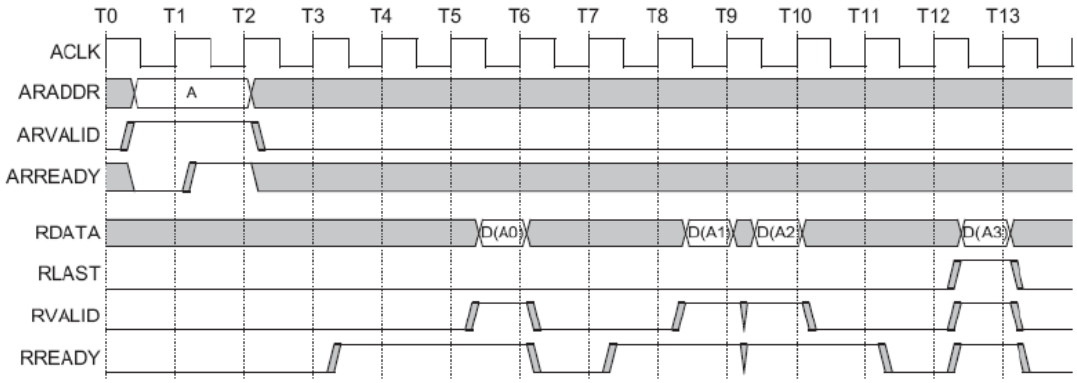


Figure 7: **Core Read Ports**



The Soft ECC Proxy LogiCORE IP uses the industry standard AMBA AXI4 Protocol Specification.

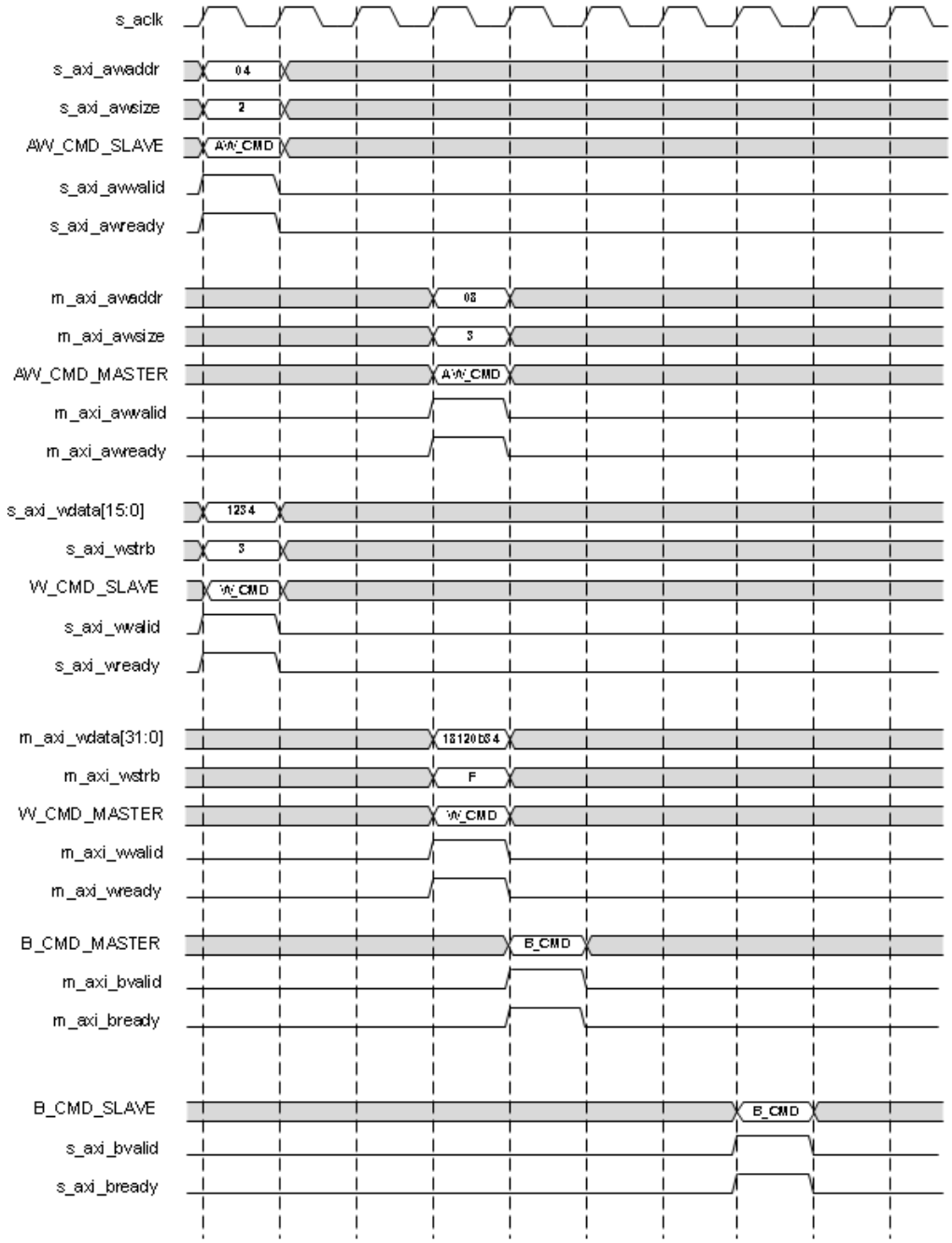
- **AXI register Slice:** The handshake mechanism from slave AXI to Master AXI is implemented with Register Slice for each channel.
- **Address Converter:** The ECC value is calculated for every byte of input data separately and placed along with data bits. Each data byte concatenated with its calculated check bits (8-bits) (check bits are in MSB) and become 16-bits. Thus, N-bytes of data becomes 2N-bytes after encoding. The AXI4 address is byte addressable. The address value and address width can be converted to handle the doubling of data width after encoding. Ex: Address value 0x40 on slave AXI interface converted to 0x80 on master AXI interface for the same data byte.
- **Error Injection:** The Soft ECC Proxy LogiCORE IP supports error injection feature and it can be enabled using C_EN_ERROR_INJECT parameter. When this feature in enabled, a single/ double bit error is injected on to every byte of write data after encoding if the corresponding input signal(inject_sbiterr/dbiterr) is high. If both the input signals are high, then double bit error is injected.

  Note: Note: One single bit error is injected for every 16bits of data after encoding. One double bit error is injected for every 16 bits of data after encoding(one on actual data byte and one on ecc byte). The position of the error changes randomly.

- **Read Outstanding:** The Soft ECC Proxy LogiCORE IP supports read outstanding feature and it can be enabled using C_OUTSTANDING_SUPPORT_RD parameter. This parameter ranges from 1 to 64. The value of 1 means read outstanding is not supported. The value of greater than 1 means read outstanding is supported. It means these many number of read transactions can be stored in the Read address channel AXI register slice when the master is not ready to accept the read transactions.

The following figures detail the AXI4 Write burst transaction and the AXI4 Read burst transaction respectively.

Figure 8: **AXI4 Write Burst Transaction**

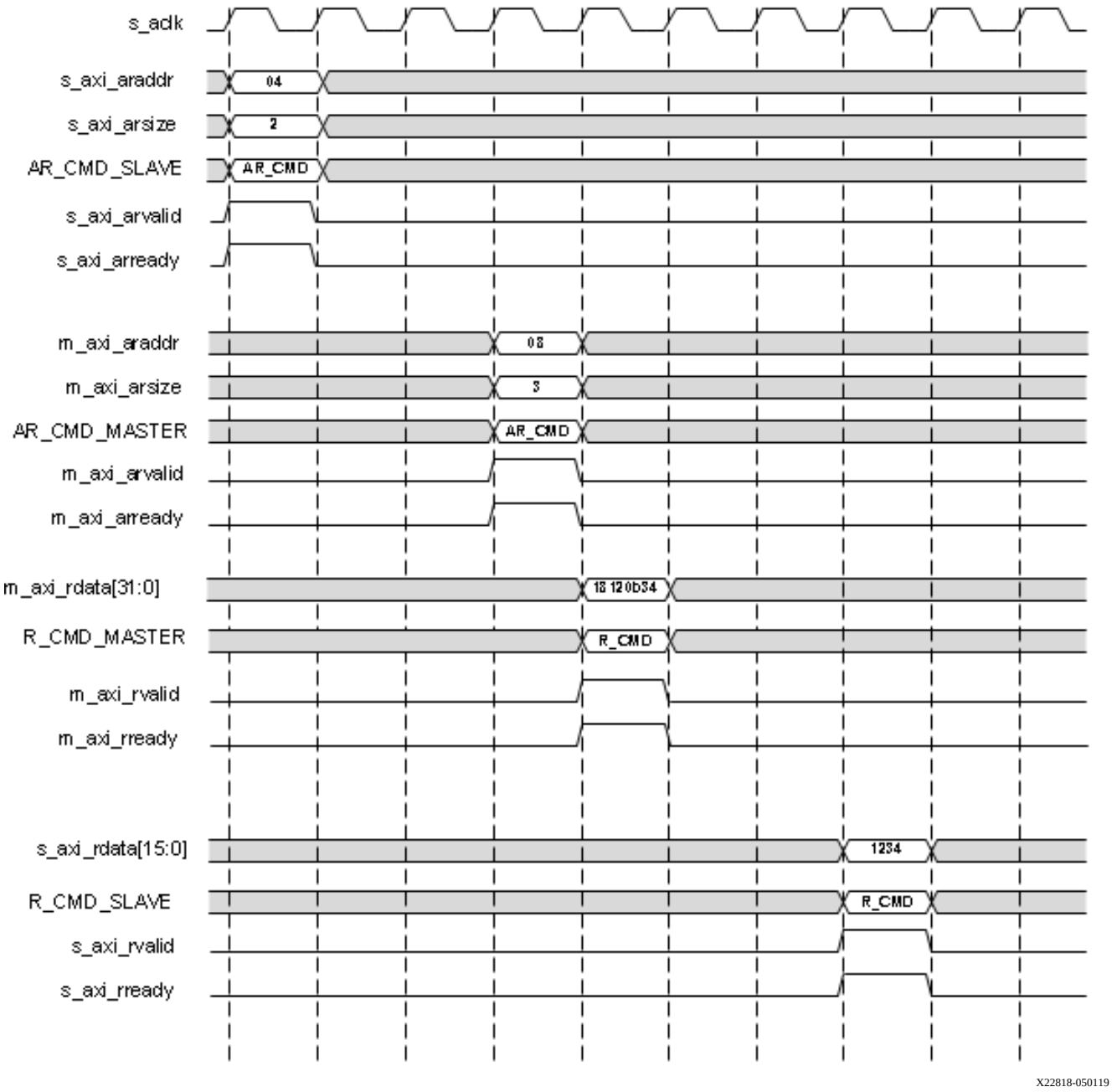


From the figure:

- **AW_CMD_SLAVE:** Write Address command signals on slave interface other than `awaddr`, `awsize`, `awvalid`, and `awready`.
- **W_CMD_SLAVE:** Write Data interface signals on slave interface other than `wdata`, `wstrb`, `wvalid`, and `wready`.

- **B_CMD_SLAVE:** Write Response channel signals on slave interface other than `bvalid` and `bready`.
- **AW_CMD_MASTER:** Write Burst Write Address command signals on master interface other than `awaddr`, `awsize`, `awvalid`, and `awready`.
- **W_CMD_MASTER:** Write Data interface signals on master interface other than `wdata`, `wstrb`, `wvalid`, and `wready`.
- **B_CMD_MASTER:** Write Response channel signals on master interface other than `bvalid` and `bready`.

Figure 9: **AXI4 Read Burst Transaction**



From the figure:

- **AR_CMD_SLAVE:** Read Address command signals on slave interface other than `araddr`, `arsize`, `arvalid`, and `arready`.
- **R_CMD_SLAVE:** Read Data interface signals on slave interface other than `rdata`, `rvalid`, and `rready`.

- **AR_CMD_MASTER:** Read Address command signals on master interface other than `araddr`, `arsize`, `arvalid`, and `arready`.
- **R_CMD_MASTER:** Read Data interface signals on master interface other than `rdata`, `rvalid`, and `rready`.

ECC Encoder

The ECC Encoder is used to generate the ECC check (or protection) bits for the input data (Write data of the AXI4 slave). For each 8-bits of data input, it generates 5 check bits. The hamming/HSIAO algorithm is used to generate check bits inside the ECC Encoder. These bits are used during each ECC Decoder operation to correct any single-bit errors, or to detect any double-bit errors. The data along with calculated check bits are provided as an output of ECC Encoder block. For details on Hamming/HSIAO algorithm, see *ECC LogiCORE IP Product Guide* (PG092).

The following is the block diagram of the ECC Encoder:

Figure 10: **ECC Encoder**

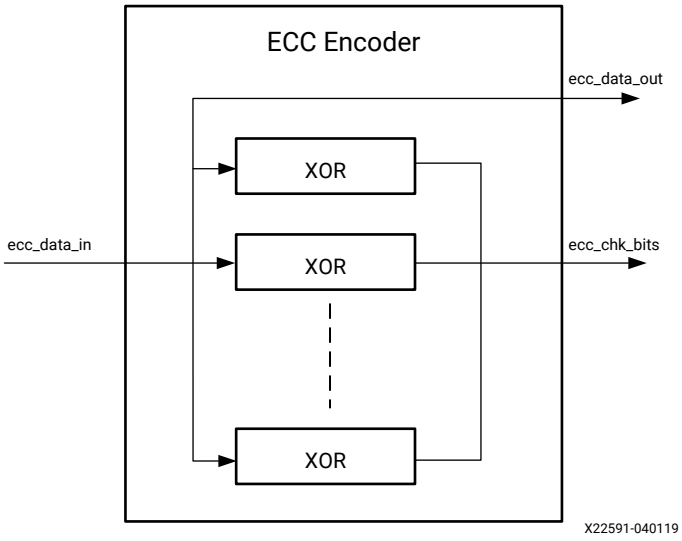
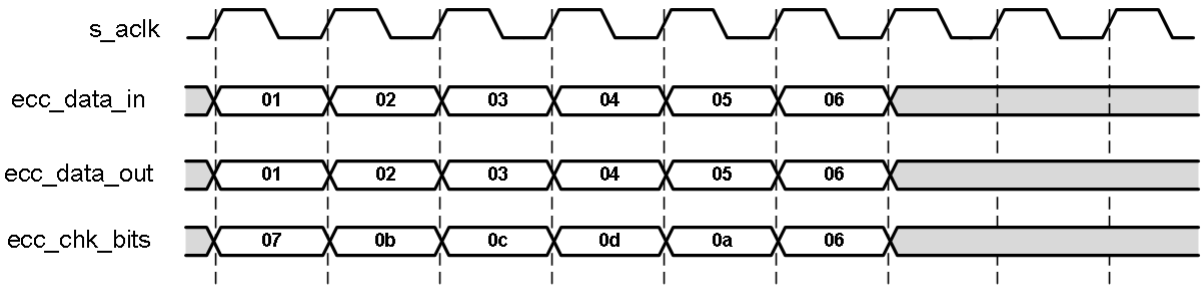


Figure 11: **ECC Encoder Waveform**



ECC Decoder

The ECC Decoder generates error correction syndrome bits for the input data and check bits. The hamming/HSIAO algorithm is used to generate syndrome bits inside ECC Decoder. These syndrome bits are used to correct any single-bit errors, or to detect (but not correct) any double-bit errors in the input data. The single-bit error corrected data, along with the `ecc_sbit_err` status or the uncorrected data with `ecc_dbit_err` status, are provided as an output of the ECC Decoder function. For details on Hamming/HSIAO algorithm, see *ECC LogiCORE IP Product Guide* (PG092).

The following is the block diagram of the ECC Decoder:

Figure 12: **ECC Decoder**

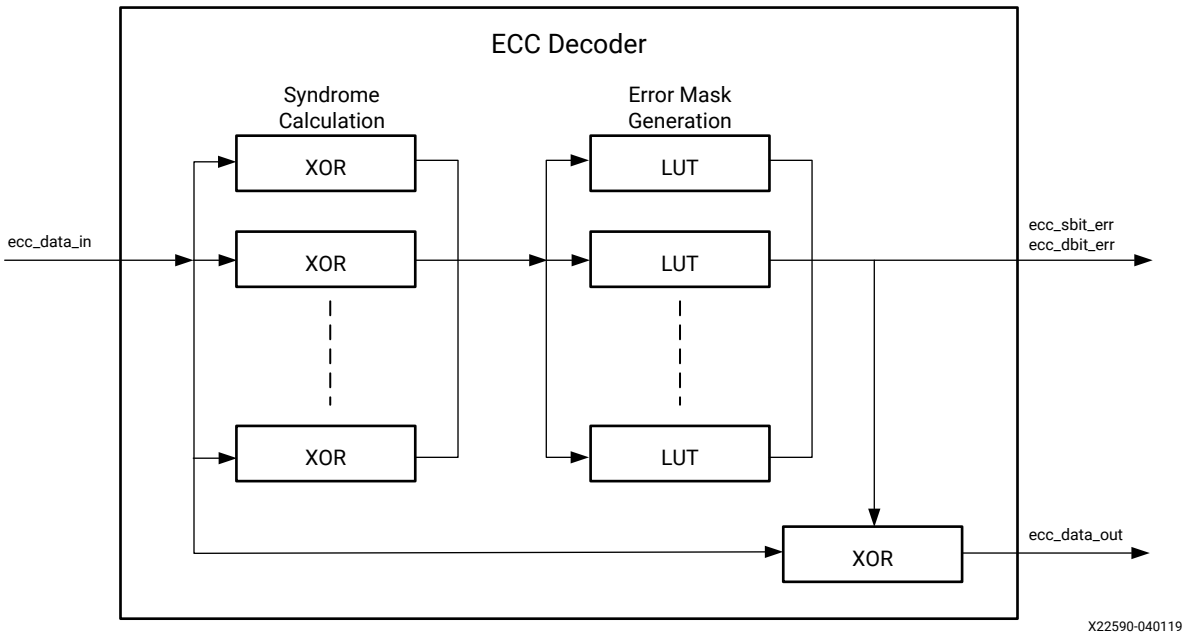
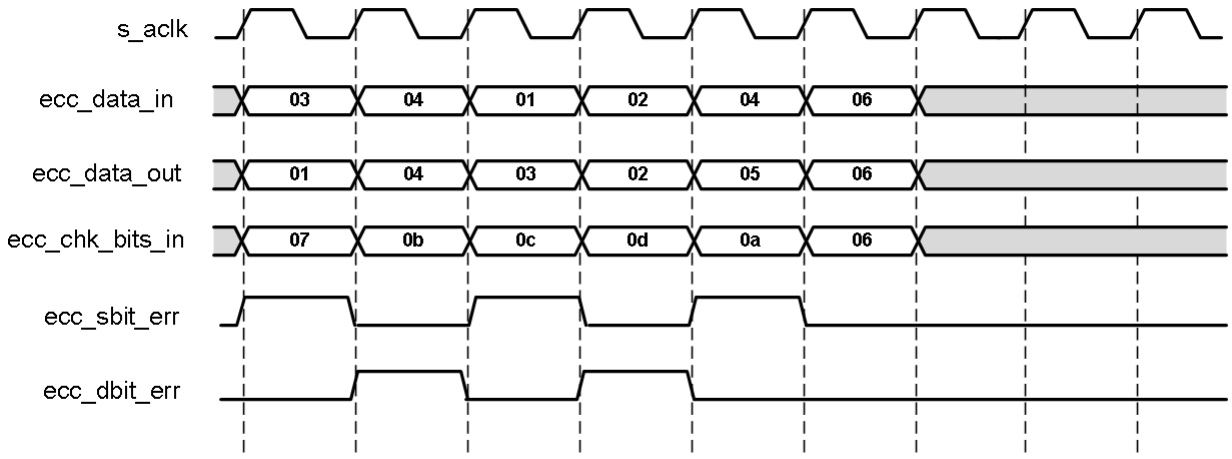


Figure 13: **ECC Decoder Waveform**



If no errors are detected, the input data is forwarded at the output, and both `ecc_sbit_err` and `ecc_dbit_err` status outputs are kept de-asserted.

The OR operation of the error signals (`ecc_sbit/dbit_err`) is connected to Soft ECC Proxy output `ecc_interrupt`.

Design Flow Steps

This section describes customizing and generating the core, constraining the core, and the simulation, synthesis, and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

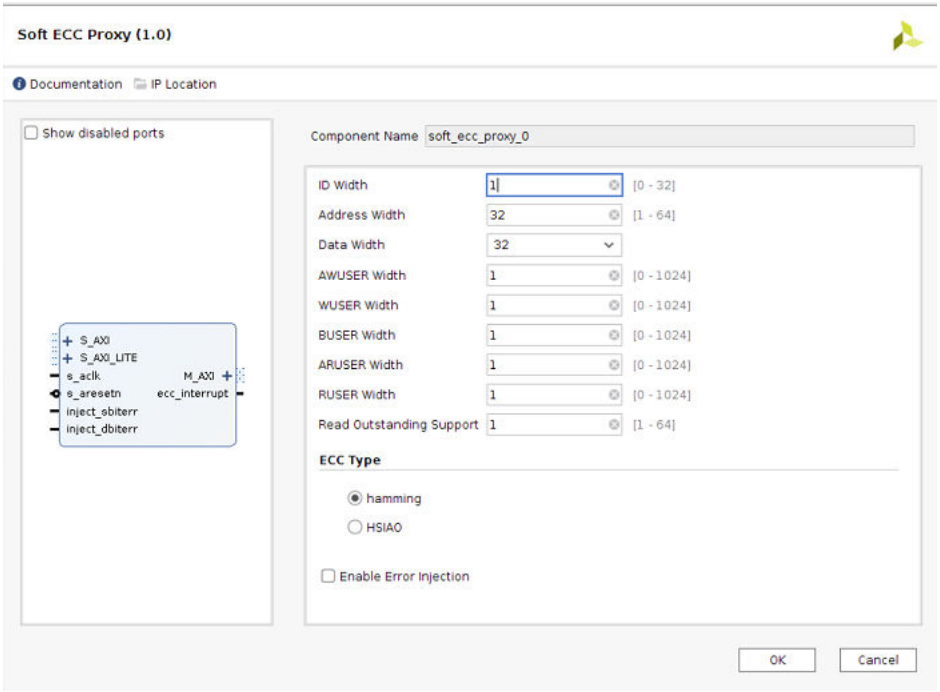
1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

Soft ECC Proxy Basic Options

Figure 14: **Soft ECC Vivado Options**



- **ID Width:** AXI4 ID Width. Supported Range: 0 to 32.
- **Address Width:** AXI4 Address Width on Slave Interface. Supported range: 1 to 64.
- **Data Width:** AXI4 Data Width on Slave Interface. Supported Values: 8, 16, 32, 64, 128, 256, and 512.
- **AWUSER Width:** AXI4 Write Address Channel User. Supported Range: 0 to 1024.
- **WUSER Width:** AXI4 Write Data Channel User. Supported Range: 0 to 1024.
- **BUSER Width:** AXI4 Write Response Channel User. Supported Range: 0 to 1024.
- **ARUSER Width:** AXI4 Read Address Channel User. Supported Range: 0 to 1024.
- **RUSER Width:** AXI4 Read Data Channel User. Supported Range: 0 to 1024.
- **Read Outstanding Support:** Number of Outstanding Addresses supported on Read Address Channel. Supported Range: 1 to 64.
- **ECC Type:** Type of Algorithm used for ECC encoding and decoding. Supported Values: Hamming, HSIAO.
- **Enable Error Injection:** Enables Error Injection feature.

User Parameters

The following table shows the relationship between the fields in the Vivado® IDE and the user parameters (which can be viewed in the Tcl Console).

Table 14: **User Parameters**

Vivado IDE Parameter/Value[1]	User Parameter/Value	Default Value
ID Width	AXI_ID_WIDTH	1
Address Width	AXI_ADDR_WIDTH	32
Data Width	AXI_DATA_WIDTH	32
AWUSER Width	AXI_AWUSER_WIDTH	1
WUSER Width	AXI_WUSER_WIDTH	1
BUSER Width	AXI_BUSER_WIDTH	1
ARUSER Width	AXI_ARUSER_WIDTH	1
RUSER Width	AXI_RUSER_WIDTH	1
Read Outstanding Support	OUTSTANDING_SUPPORT_RD	1
ECC Type	ECC_TYPE	Hamming
Enable Error Injection	EN_ERROR_INJECT	False

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Constraining the Core

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

The maximum frequencies that the core can work with default configurations is 350 MHz.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado® simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

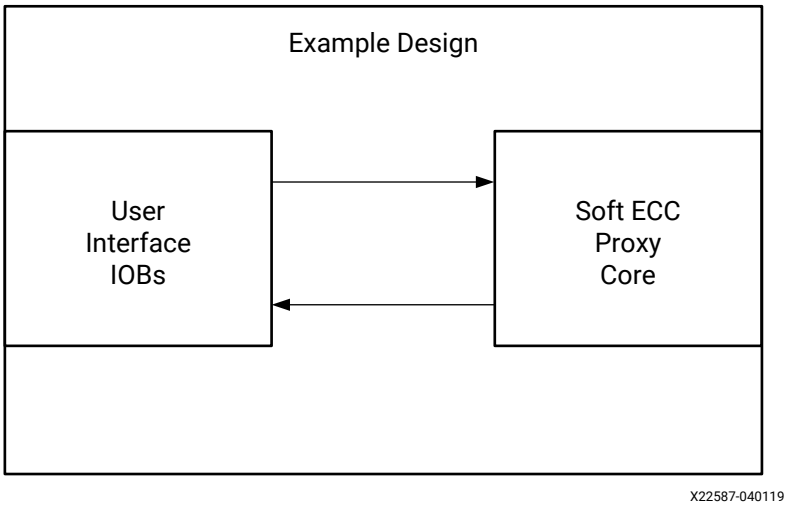
Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Example Design

The following figure shows the example design for the Soft ECC Proxy LogiCORE IP.

Figure 16: **Core Example Design**



The example design contains the following:

- An instance of the Soft ECC Proxy LogiCORE IP. During simulation, the Soft ECC Proxy core is instantiated as a black box and replaced during implementation with the structural netlist model generated by the Vivado® IP Catalog IP customizer for timing simulation or a behavioral model for the functional simulation.
- Global clock buffers for top-level port clock signals.

Implementing the Example Design

After generating a core, right click on the generated core and click **Open IP Example Design**. In the example project tab, click the **Run Synthesis and Run Implementation** option to implement the example design.

Simulating the Example Design

The Soft ECC Proxy core provides a quick way to simulate and observe the behavior of the core by using the provided example design. There are five different simulation types:

- Behavioral
- Post-Synthesis Functional
- Post-Synthesis Timing
- Post-Implementation Functional
- Post-Implementation Timing

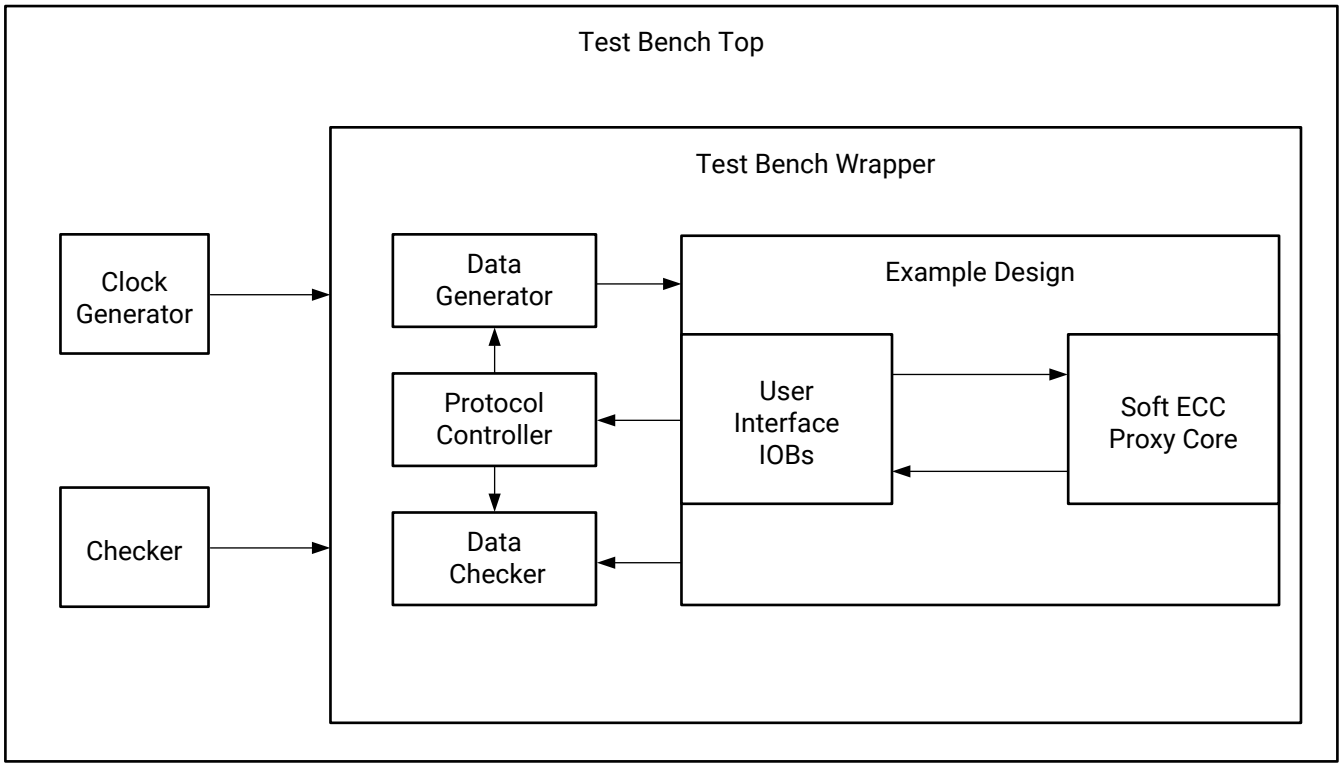
Chapter 7

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

The following figure shows a block diagram of the subsystem test bench.

Figure 18: **Soft ECC Proxy Test Bench**



Test Bench Functionality

The demonstration test bench is a straightforward Verilog-HDL file that can be used to exercise the example design and the core itself. The test bench consists of the following:

- Clock Generators

- Data generator module
- Data verifier module
- Module to control data generator and verifier

The demonstration test bench in a core with an AXI interface performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- Pseudo random data is generated and given as input to AXI Interface input signals. Each channel is independently checked for Valid-Ready handshake protocol.
- AXI output signals on read side are combined and cross checked with the pseudo random generator data.
- For AXI memory mapped interface five instances of data generator, data verifier and protocol controller are used.

Messages and Warnings

When the functional or timing simulation has completed successfully, the test bench displays the following message, and it is safe to ignore this message.

Failure: Test Completed Successfully

Upgrading

This appendix is not applicable for the first release of the core.

Appendix B

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT! *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Finding Help on Xilinx.com

To help in the design and debug process when using the core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The Xilinx Community Forums are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx® Documentation Navigator. Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Technical Support

Xilinx provides technical support on the Xilinx Community Forums for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To ask questions, navigate to the Xilinx Community Forums.

Debug Tools

There are many tools available to address Soft ECC Proxy design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx® devices.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

***Note*:** For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. R. W. Hamming. 1950. "Error Detecting and Error Correcting Codes," Bell System Technical Journal.
2. M. Y. Hsiao. 1970. "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," IBM Journal of Research and Development.
3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
4. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
5. *Vivado Design Suite User Guide: Implementation* (UG904)
6. *Vivado Design Suite User Guide: Designing with IP* (UG896)
7. *ISE to Vivado Design Suite Migration Guide* (UG911)
8. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
9. *Vivado Design Suite User Guide: Getting Started* (UG910)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
08/13/2021 Version 1.0	
Document title	Changed title to Soft ECC Proxy LogiCORE IP Product Guide
07/14/2020 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

This document contains preliminary information and is subject to change without notice. Information provided herein relates to products and/or services not yet available for sale, and provided solely for information purposes and are not intended, or to be construed, as an offer for sale or an attempted commercialization of the products and/or services referred to herein.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright