HDCP 1.x v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG224 December 20, 2019

Σ XILINX

Table of Contents

Introduction

The Xilinx® LogiCORE™ IP High-bandwidth Digital Content Protection (HDCP™) 1.3 and 1.4 protocol IP is designed for transmission of audiovisual content securely between two devices that are HDCP-capable. Data can be transmitted over a DisplayPort (DP) interface or HDMI™ interface. This IP supports HDCP 1.3 for content protection over DisplayPort and HDCP 1.4 over HDMI.

IMPORTANT: *The HDCP IP is only available as a part of the subsystems.*

Features

- Subsystem-based IP, facilitates easy integration
- HDCP 1.4 (encrypt and decrypt) for HDMI
- HDCP 1.3 (encrypt and decrypt) for DP
- Audio and Video support
- AXI4-Stream based Data interface to DisplayPort controller/HDMI controller
- Supports up to 4k × 2k at 60 frames per second
- AXI4-Stream based Key Management interface
- 1/2/4 lanes support (DisplayPort HDCP)
- SST-based encryption/decryption (DisplayPort HDCP)

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	UltraScale+™ Families, UltraScale™ Architecture, 7 Series FPGAs
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	See Table 2-2 and Table 2-3.
Provided with Core	
Design Files	Encrypted RTL
Example Design	Provided as a part of HDMI or DisplayPort example design
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Not Provided
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Xilinx Support web page	
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775

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.

Send Feedback

Overview

This chapter contains an overview of the core as well as details about applications, licensing, and standards.

The HDCP IP can be used in two variants:

- Content protection of DisplayPort packed audio video data

- Content protection of HDMI™ audio video data

The following sections give an overview of each option.

HDCP for DisplayPort (HDCP 1.3)

In this mode, the HDCP controller is used to interface with the Xilinx® DisplayPort Source/ Sink controller. Further, the controller is used for encryption or decryption depending on interfacing with DisplayPort source controller or DisplayPort sink controller, respectively.
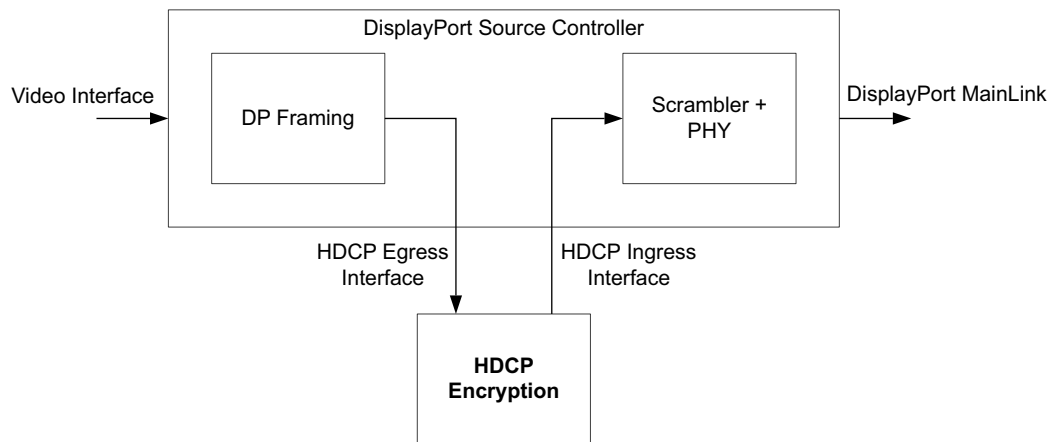
X14915-061719

Figure 1-1:   **DisplayPort Source Controller with HDCP Encryption Subsystem**

An HDCP 1.3 encryption IP can be generated using the Xilinx DisplayPort Transmit subsystem (DP TX subsystem). For more information, see the *DisplayPort TX Subsystem Product Guide* (PG199) [Ref 1].

Send Feedback

Figure 1-2: **DisplayPort Sink Controller with HDCP Decryption Subsystem**

An HDCP 1.3 decryption IP can be generated using the Xilinx DisplayPort Receive subsystem (DP RX Subsystem). For more information, see the *DisplayPort RX Subsystem Product Guide* (PG233) [Ref 2].

See the DisplayPort HDCP reference design for typical usage of the HDCP IP with the DisplayPort Source/Sink controller. For more information on the HDCP Egress and Ingress interface, see the Port Descriptions.
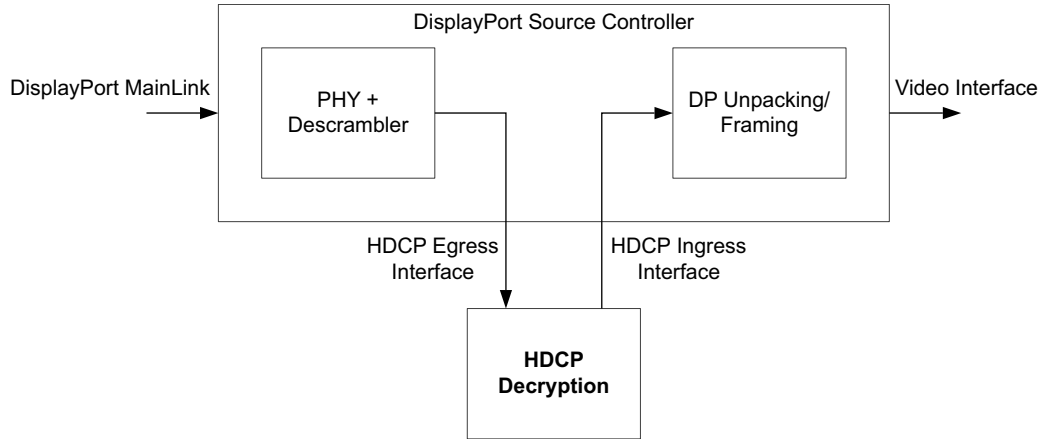
HDCP for HDMI (HDCP 1.4)

In this mode, the HDCP controller is used to interface with the HDMI controller. Further, the controller is used for encryption or decryption depending on interfacing with the HDMI TX controller or HDMI RX controller, respectively.

Figure 1-3: **HDMI TX Controller with HDCP Encryption Subsystem**

Send Feedback

An HDCP 1.4 encryption IP can be generated using the Xilinx HDMI Transmit subsystem (HDMI TX subsystem). For more information, see the *HDMI 1.4/2.0 TX Subsystem Product Guide* (PG235) [Ref 3].

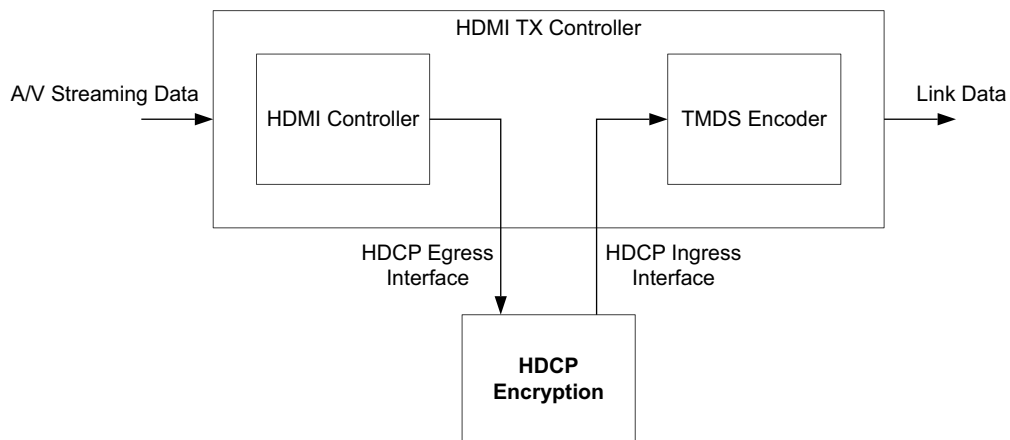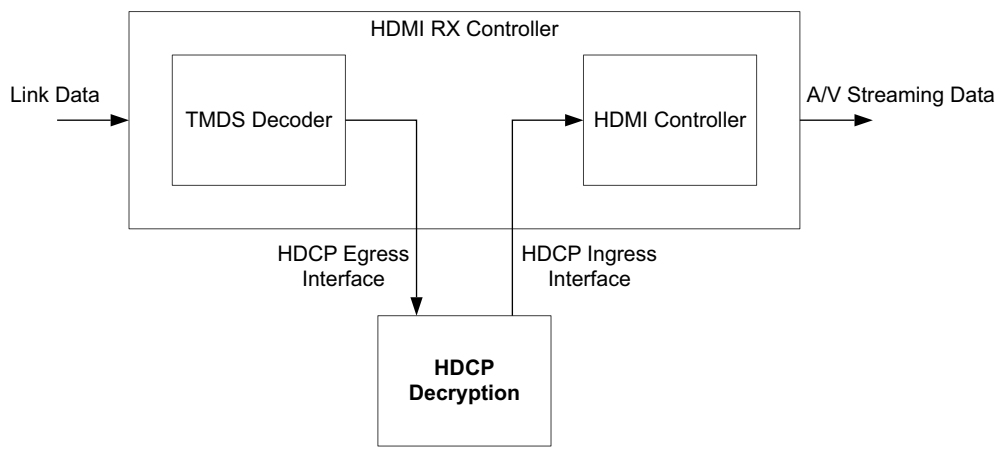*Figure 1-4:* **HDMI RX Controller with HDCP Decryption Subsystem**

An HDCP 1.4 decryption IP can be generated using the Xilinx HDMI Receive subsystem (HDMI RX subsystem). For more information, see the *HDMI 1.4/2.0 RX Subsystem Product Guide* (PG236) [Ref 4].

See the HDMI HDCP reference design for typical usage of the HDCP IP with the HDMI Source/Sink controller. For more information on the HDCP Egress and Ingress interface, see the Port Descriptions.

Feature Summary

Xilinx HDCP IP offers HDCP encryption and decryption functionality, in conjunction with DisplayPort/HDMI controller. It also provides an AXI4-Stream interface to get decrypted HDCP keys. Video resolutions up to 7680 × 4320 at 30 frames per second is supported.

For HDCP in the DisplayPort mode, single stream video and audio are supported.

HDCP is supported only as a sub-core in DisplayPort/HDMI Transmit/Receive subsystems.

Unsupported Features

•   DisplayPort MST mode encryption/decryption is not supported

•   Repeater is not supported for DisplayPort

Send Feedback

- AVMUTE, Advanced Cipher, and Enhanced Link Verification are not supported for the HDMI HDCP Controller

Licensing and Ordering Information

This Xilinx LogiCORE™ HDCP IP module is provided under the terms of the Xilinx Core License Agreement. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability of the Xilinx LogiCORE HDCP IP.

The IP/reference design(s) enables Xilinx customers to implement High-bandwidth Digital Content Protection (HDCP) cipher and authentication functions in Xilinx silicon devices using Xilinx's LogiCORE DisplayPort or High-Definition Multimedia Interface (HDMI) IP solutions. The reference design(s) which also includes logic blocks which may be used by customers at their option to securely read HDCP device keys from an external storage device and store them on a Xilinx silicon device; alternatively, customers may independently develop such logic to perform these functions. HDCP device keys are not provided with the reference design(s) and are not available from Xilinx under any circumstances. Customers who desire to utilize the IP and reference design(s) to implement HDCP must become an HDCP Adopter and acquire device keys directly from Digital Content Protection, LLC. Failure by customers to do so will render the IP and reference design(s) incapable of successfully completing the HDCP implementation in customers' products.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado® design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following 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.*

Note: Xilinx does not provide evaluation license, because it is only provided to customers who are HDCP adopters. Customers who are HDCP adopters need to contact their FAE to get the license.

Product Specification

The Xilinx® LogiCORE™ IP High-bandwidth Digital Content Protection (HDCP™) protocol is designed for transmission of audiovisual content securely between two devices that are HDCP capable. The audio visual content can be transmitted over a DisplayPort interface or HDMI™ interface. HDCP protocol specifies rules for content protection over DisplayPort (HDCP 1.3) or HDMI (HDCP 1.4).

Standards

The HDCP is designed to be compatible with High-bandwidth Digital Content Protection system Revision 1.4 and High-bandwidth Digital Content Protection system Revision 1.3 Amendment for DisplayPort. For silicon status, check the Vivado® IP catalog.

Performance

This section contains the details about the performance of this core.

Maximum Frequencies

The core uses two clock domains. For more information about these clock domains, see the Clocking in Chapter 3.

Table 2-1 shows the clock ranges.

Table 2-1:   **Maximum Frequencies**

Clock Domain	Min (MHz)	Max (MHz)	Description
hdcp_clk	18.5625	202 (DisplayPort) 297 (HDMI 1.4/2.0)	Egress interface clock/ingress interface clock
s_axi_aclk	50	200	Host processor clock

Resource Utilization

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

Port Descriptions

This section describes the HDCP IP core ports.

Table 2-2: **HDCP I/O Signals**

Signal Name	Direction with Respect to Core	Description
Host Interface (AXI4-Lite)		
s_axi_aclk	Input	AXI4-Lite Clock
s_axi_aresetn	Input	Active-Low AXI4-Lite Reset
s_axi_awaddr [7:0]	Input	Write Address
s_axi_awvalid	Input	Write Address Valid
s_axi_awready	Output	Write Address Ready
s_axi_wdata [31:0]	Input	Write Data
s_axi_wstrb [3:0]	Input	Write Strobe
s_axi_wvalid	Input	Write Valid
s_axi_wready	Output	Write Ready
s_axi_bready	Input	Response Ready
s_axi_bresp[1:0]	Output	Write Response
s_axi_bvalid	Output	Write Response Valid
s_axi_araddr [7:0]	Input	Read Address
s_axi_arvalid	Input	Read Address Valid
s_axi_arready	Output	Read Address Ready
s_axi_rready	Input	Read Ready
s_axi_rdata[31:0]	Output	Read Data
s_axi_rresp[1:0]	Output	Read Response
s_axi_rvalid	Output	Read Valid
irq	Output	Interrupt
Streaming Interface		
hdcp_clk	Input	HDCP Clock
hdcp_clken	Input	Clock Enable
hdcp_rst	Input	HDCP Reset

Table 2-2: **HDCP I/O Signals** *(Cont'd)*

Signal Name	Direction with Respect to Core	Description
Egress Data (AXI4-Stream Slave)		
hdcp_egress_tdata[N:0][1]	Input	Data width is 128-bit for DisplayPort and 96-bit for HDMI.
hdcp_egress_tuser[N:0][2]	Input	TUSER width is 16-bit for DisplayPort and 32-bit for HDMI.
hdcp_egress_tstrb[3:0]	Input	Egress Strobe
hdcp_egress_tid	Input	Egress TID
hdcp_egress_tlast	Input	Egress TLAST
hdcp_egress_tready	Output	Egress TREADY
hdcp_egress_tvalid	Input	Egress TVALID
Ingress Data (AXI4-Stream Master)		
hdcp_ingress_tdata[127:0][1]	Output	Data width is 128-bit for DisplayPort and 96-bit for HDMI.
hdcp_ingress_tuser [31:0][2]	Output	TUSER width is 16-bit for DisplayPort and 32-bit for HDMI. For more information on TUSER bits for HDCP 1.3 and HDCP 1.4, see Table 2-3 and Table 2-4, respectively.
hdcp_ingress_tstrb[3:0]	Output	Ingress Strobe
hdcp_ingress_tid	Output	Ingress TID
hdcp_ingress_tlast	Output	Ingress TLAST
hdcp_ingress_tready	Input	Ingress TREADY
hdcp_ingress_tvalid	Output	Ingress TVALID
HDCP Key Management Block Interface (AXI4-Stream Slave)		
hdcp_key_aclk	Output	Key Clock
hdcp_key_aresetn	Output	Key interface reset. Active-Low.
hdcp_key_tdata[63:0]	Input	AXI4-Stream Key Data
hdcp_key_tuser[7:0]	Input	AXI4-Stream Key TUSER. KMB should send the Key number from 0 to 41. 0 corresponds to KSV and 1 to 40 are the HDCP Keys count.
hdcp_key_tready	Output	AXI4-Stream Key Ready
hdcp_key_tlast	Input	AXI4-Stream Key TLAST
hdcp_key_tvalid	Input	AXI4-Stream Key TVALID
start_key_transmit	Output	An Active-High pulse that is used to start key transmit.

Table 2-2: **HDCP I/O Signals** *(Cont'd)*

Signal Name	Direction with Respect to Core	Description
reg_key_sel[2:0]	Output	To select the one of the eight sets of 40 keys.

Notes:

1. TDATA width depends on the protocol. For HDCP 1.3 (DisplayPort), TDATA width is 128 bits. For HDCP 1.4 (HDMI), TDATA width is 96 bits.
2. TUSER width depends on the protocol. For HDCP 1.3 (DisplayPort), TUSER width is 16 bits. For HDCP 1.4 (HDMI), TUSER width is 32 bits.

Table 2-3: **HDCP 1.3 (DisplayPort) TUSER**

Signal Name	Width	Direction from Video Controller	Clock	Description
hdcp_egress_tuser	32	Output	hdcp_clk	K-Char indicator in TUSER field. Bits[15:12] = Lane 3 K-char Bits[11:8] = Lane 2 K-char Bits[7:4] = Lane 1 K-char Bits[3:0] = Lane 0 K-char

Table 2-4: **HDCP 1.4 (HDMI) TUSER**

Signal Name	Width	Direction from Video Controller	Clock	Description
hdcp_egress_tuser	32	Output	hdcp_clk	Bits[31:28] = Keep out region indicator (debug only) Bits[27:24] = Window of opportunity indicator Bits[23:20] = EESS control signaling (lane 3) Bit[19] = Vertical sync (lane 3). Bit[18] = Horizontal sync (lane 3) Bits[17:14] = EESS control signaling (lane 2) Bits[13] = Vertical sync (lane 2) Bit[12] = Horizontal sync (lane 2) Bits[11:8] = EESS control signaling (lane 1) Bit[7] = Vertical sync (lane 1) Bit[6] = Horizontal sync (lane 1) Bits[5:2] = EESS control signaling (lane 0) Bit[1] = Vertical sync (lane 0) Bit[0] = Horizontal sync (lane 0)

Register Space

Table 2-5 shows the register map for DisplayPort and HDMI.

Table 2-5: **DisplayPort and HDMI HDCP Register Map**

Offset	Name	Access	Description
DisplayPort			
0x0000	Version	RO	
0x0004	Type	RO	Bits[1:0] = Protocol (00 = DP, 01 = HDMI, 10/11 = RFFU) Bit[2] = Direction (0 = RX, 1 = TX) Bit[3] = Mode (0 = SST, 1 = MST) (to be confirmed)
0x0008	Reserved	RSVD	Reserved
0x000C	Control	RW	Bit[0] = Enable (0 = Bypass, 1 = Enabled) Default = 0 Bit[1] = Register Update (0 = Disabled, 1 = Update Immediately) Bit[3] = Mode (0 = SST, 1 = MST) (to be confirmed) Bits[6:4] = Number of lanes (001 = 1, 010 = 2, 100 = 4) Default = 100 Bit[31] = Reset
0x0010	Status	RO	Bit[0] = Link Integrity Failure Bit[1] = Ri Update Complete
0x0014	Interrupt Enable	RW	Bit[0] = Link Integrity Failure Bit[1] = Ri Update Complete
0x0018	Interrupt Status	RO/W1C	Bit[0] = Link Integrity Failure Bit[1] = Ri Update Complete
0x001C	Reserved	RSVD	Reserved
0x0020	Encryption Control (H)	RW (RO in RX)	Bits[30:0] = Encrypt Signalling[62:32]
0x0024	Encryption Control (L)	RW (RO in RX)	Bits[31:0] = Encrypt Signalling[31:0]
0x002C	Key Mgmt Block Control	RW	Bit[0] = Read Local KSV. Set this to 0 when KSV value is available. Bit[1] = Begin Km calculation. Set this to 0 when Km value is available. Bit[2] = Abort Km calculation. Toggle this bit to 1 and then to 0 for abort of Km calculation. Bits[18:16] = Key Management Set Select (000-111) Default = 000
0x0030	Key Mgmt Block Status	RO	Bit[0] = Local KSV available Bit[1] = Km value available
0x0034	Reserved	RSVD	Reserved
0x0038	Local KSV (H)	RO	Bits[7:0] = Local KSV[39:32]
0x003C	Local KSV (L)	RO	Bits[31:0] = Local KSV[31:0]
0x0040	Remote KSV (H)	RW	Bits[7:0] = Remote KSV[39:32]
0x0044	Remote KSV (L)	RW	Bits[31:0] = Remote KSV[31:0]
0x0048 to 0x004C	Reserved	RSVD	Reserved

HDCP 1.x v1.0
PG224 December 20, 2019   www.xilinx.com   **13**
Send Feedback

Table 2-5: **DisplayPort and HDMI HDCP Register Map** *(Cont'd)*

Offset	Name	Access	Description
0x0050	Cipher Control	RW	Bit[0] = XOR Enable Bit[8] = Initiate Block Cipher Bit[9] = Initiate ReKey Cipher Bit[10] = Initiate RNG Cipher
0x0054	Cipher Status	RO	Bit[0] = XOR In Progress Bit[8] = Block Cipher In Progress Bit[9] = ReKey Cipher In Progress Bit[10] = RNG Cipher In Progress Bit[11] = Mi Update In Progress Bit[12] = Ri Update In Progress
0x0058	Cipher Bx	RW	Bits[27:0] = Value Bits[27:0]
0x005C	Cipher By	RW	Bits[27:0] = Value Bits[55:28]
0x0060	Cipher Bz	RW	Bits[27:0] = Value Bits[63:56]
0x0078	Cipher Ri	RO	Bits[15:0] = Ri register
0x007C	Cipher Ro	RO	Bits[15:0] = Ro register (from last completed block cipher request)
0x0080	Cipher Mo	RO	M0 value upper 32-bits register
0x0084	Cipher Mo	RO	M0 value lower 32-bits register
HDMI			
0x0000	Version	RO	
0x0004	Type	RO	Bits[1:0] = Protocol (00 = DP, 01 = HDMI, 10/11 = RFFU) Bit[2] = Direction (0 = RX, 1 = TX)
0x0008	Reserved	RSVD	Reserved
0x000C	Control	RW	Bit[0] = Enable (0 = Bypass, 1 = Enabled) Default = 0 Bit[1] = Register Update (0 = Disabled, 1 = Update Immediately) Bits[6:4] = Number of lanes (001 = 1, 010 = 2, 100 = 4) Default = 100 (DisplayPort only) Bit[31] = Reset
0x0010	Status	RO	Bit[0] = Link Integrity Failure (DisplayPort only) Bit[1] = Ri Update Complete (HDMI only)
0x0014	Interrupt Enable	RW	Bit[0] = Link Integrity Failure (DisplayPort only) Bit[1] = Ri Update Complete (HDMI only)
0x0018	Interrupt Status	RO/W1C	Bit[0] = Link Integrity Failure (DisplayPort only) Bit[1] = Ri Update Complete (HDMI only)
0x001C	Reserved	RSVD	Reserved
0x0020	Encryption Control (H)	RW (RO in RX)	Bits[30:0] = Encrypt Signalling[62:32] (DisplayPort only)
0x0024	Encryption Control (L)	RW (RO in RX)	Bits[31:0] = Encrypt Signalling[31:0] (DisplayPort only)

Send Feedback

Table 2-5: **DisplayPort and HDMI HDCP Register Map** *(Cont'd)*

Offset	Name	Access	Description
0x002C	Key Mgmt Block Control	RW	Bit[0] = Read Local KSV. Set this to 0 when KSV value is available. Bit[1] = Begin Km calculation. Set this to 0 when Km value is available. Bit[2] = Abort Km calculation. Toggle this bit to 1 and then to 0 for abort of Km calculation. Bits[18:16] = Key Management Set Select (000-111) Default = 000
0x0030	Key Mgmt Block Status	RO	Bit[0] = Local KSV available Bit[1] = Km value available
0x0034	Reserved	RSVD	Reserved
0x0038	Local KSV (H)	RO	Bits[7:0] = Local KSV[39:32]
0x003C	Local KSV (L)	RO	Bits[31:0] = Local KSV[31:0]
0x0040	Remote KSV (H)	RW	Bits[7:0] = Remote KSV[39:32]
0x0044	Remote KSV (L)	RW	Bits[31:0] = Remote KSV[31:0]
0x0048 to 0x004C	Reserved	RSVD	Reserved
0x0050	Cipher Control	RW	Bit[0] = XOR Enable Bit[8] = Initiate Block Cipher Bit[9] = Initiate ReKey Cipher Bit[10] = Initiate RNG Cipher
0x0054	Cipher Status	RO	Bit[0] = XOR In Progress Bit[8] = Block Cipher In Progress Bit[9] = ReKey Cipher In Progress Bit[10] = RNG Cipher In Progress Bit[11] = Mi Update In Progress Bit[12] = Ri Update In Progress
0x0058	Cipher Bx	RW	Bits[27:0] = Value Bits[27:0]
0x005C	Cipher By	RW	Bits[27:0] = Value Bits[55:28]
0x0060	Cipher Bz	RW	Bits[27:0] = Value Bits[63:56]
0x0078	Cipher Ri	RO	Bits[15:0] = Ri register
0x007C	Cipher Ro	RO	Bits[15:0] = Ro register (from last completed block cipher request)

Designing with the Core

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

General Design Guidelines

The architecture for the HDCP controller is given in the following sections. Key operations are LFSR, Block Module, and Cipher Logic. In Receive operation, the byte alignment logic is used to properly trigger frame key and line rekey events.

This section gives information on the HDCP Key interface. The HDCP Key interface is a standard AXI4-Stream interface. This is used to read keys from an external block. The external block is called the Key Management Block (KMB).

IMPORTANT: *It is important to note that the LogiCORE HDCP IP does not include KMB. You will need to use your own Key Management block and integrate it into your HDCP design as shown in Figure 3-1. You should also refer to the Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs (XAPP1084) [Ref 6].*

The HDCP Key interface is active only after it is enabled by asserting `start_key_transmit`, where it reads the public KSV and the 40 private keys in one shot and then the interface halts. To reread all keys again, the enable signal `start_key_transmit` should be released and applied through a Control register. Also through the key selection control bus `reg_key_sel[2:0]`, it is possible to select one of the eight different keysets. These control signals can be written to Key Management Block Control register (0x002C) Bits[18:16].
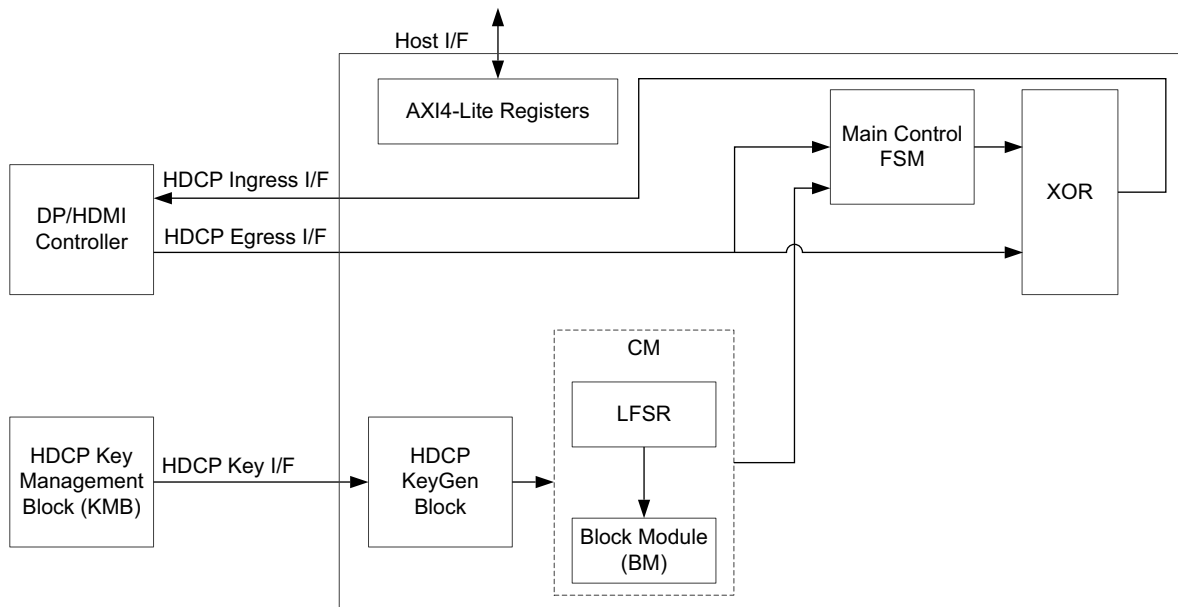
X14919-061719

Figure 3-1: **HDCP DisplayPort and HDMI IP Block Diagram**

The usage of the HDCP IP is simplified through the subsystems. The DisplayPort and HDMI™ subsystems need to be used to use HDCP for DisplayPort and HDMI, respectively.

Figure 3-2 is indicative of DisplayPort TX subsystem with the HDCP Enabled. This figure is shown only to indicate HDCP interfaces of the subsystem, see *Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs* (XAPP1048) [Ref 6]. For more information on design and interfacing DisplayPort TX subsystem, see *DisplayPort TX Subsystem Product Guide* (PG199) [Ref 1]. For details on using HDCP for DisplayPort and HDMI, see the respective reference design.

The subsystems connect the Egress and Ingress data interface of HDCP IP to the respective controller. Connect the HDCP AXI4-Lite host interface and HDCP Key interface to the Host processer and KMB, respectively.
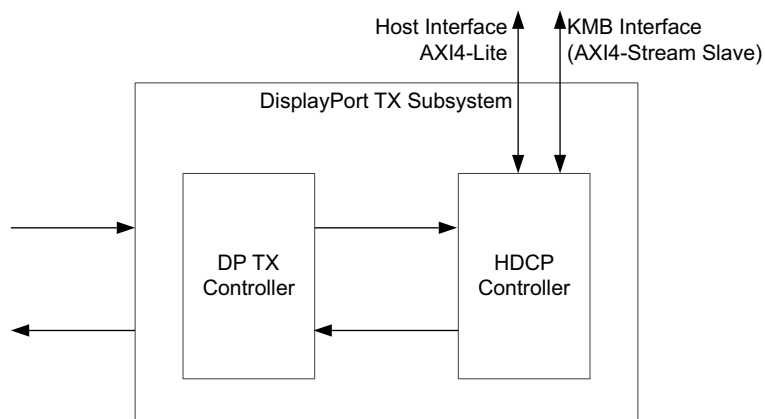
X14920-061719

Figure 3-2: **HDCP Enabled DisplayPort Transmit Subsystem**

HDCP Key Management Block

The HDCP Key Management Block (KMB) has to be designed and integrated in the design.

The HDCP KMB block should interface with the Xilinx HDCP IP over the KMB interface. This is a standard AXI4-Stream interface.

HDCP IP – KMB Interface Description

The HDCP IP expects KMB to start driving its signals according the AXI4-Stream specification. The HDCP IP asserts AXI4-Stream **TREADY** signal. The Key Management module should send the keys over its AXI4-Stream master interface, starting with address first (Ksv) and followed by the rest of the 40 private keys.

The following sequence specifies the interaction of the HDCP IP with Key Management Block. The entire key read is based on the appropriate trigger from the controlling processor. The available controls include:

1. Begin KSV Read (Write to 0x2C Bit[0]) – This initiates KSV Read from KMB. Software must ensure to prepare KMB with correct KSV before this trigger.

2. Begin Km Calculation (Write to 0x2C Bit[1]) – This initiates read of 40 HDCP Keys from the KMB. Software must ensure to prepare KMB with proper HDCP Key set before setting this trigger.

3. Abort Km calculation (Write to 0x2C Bit[2]) – This trigger orphans the current read of HDCP Keys. The HDCP IP generates a trigger to KMB **start_key_transmit**. After receiving this, KMB should abandon any Key transfer in progress. KMB should go back to idle and wait for TREADY from HDCP.
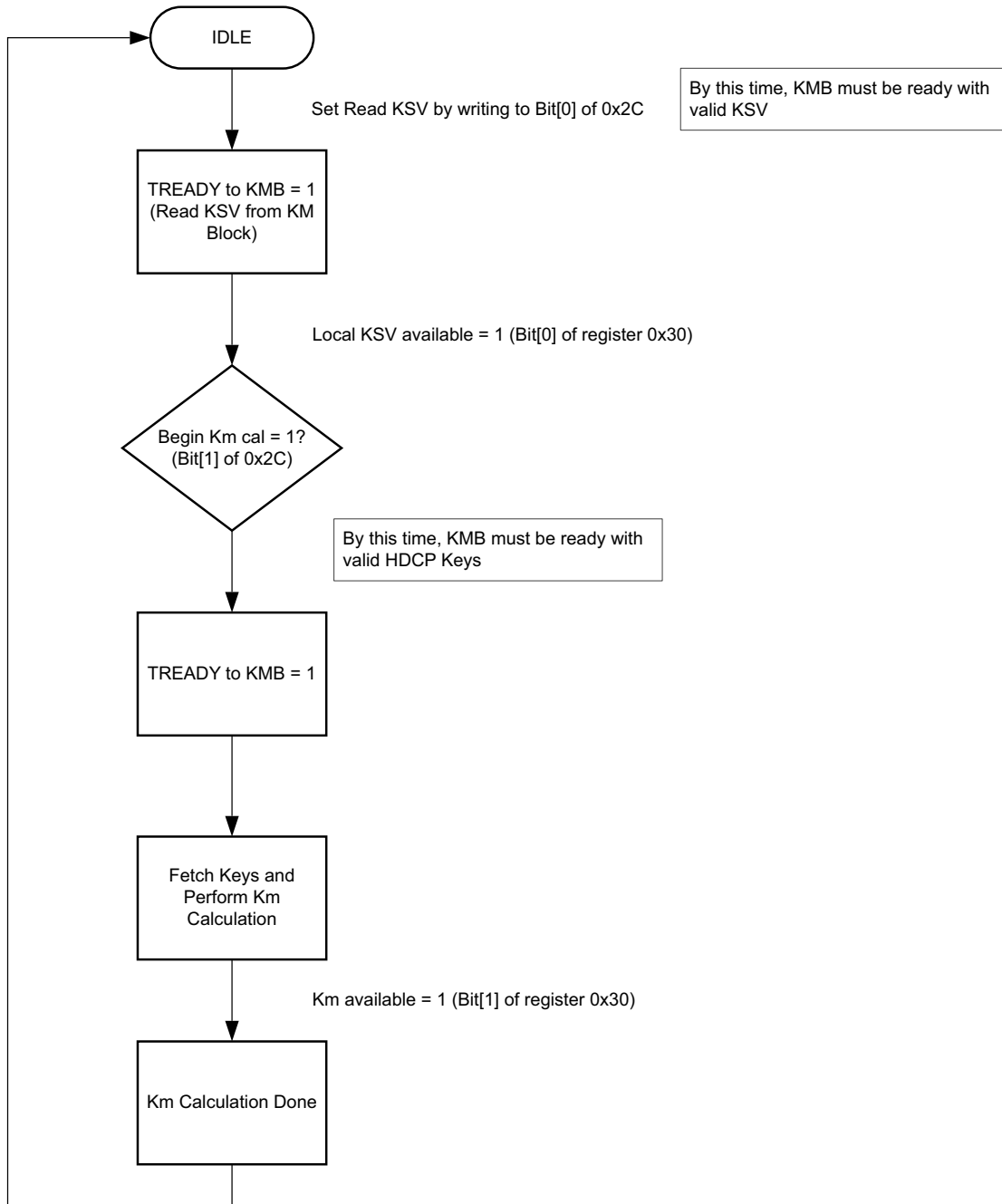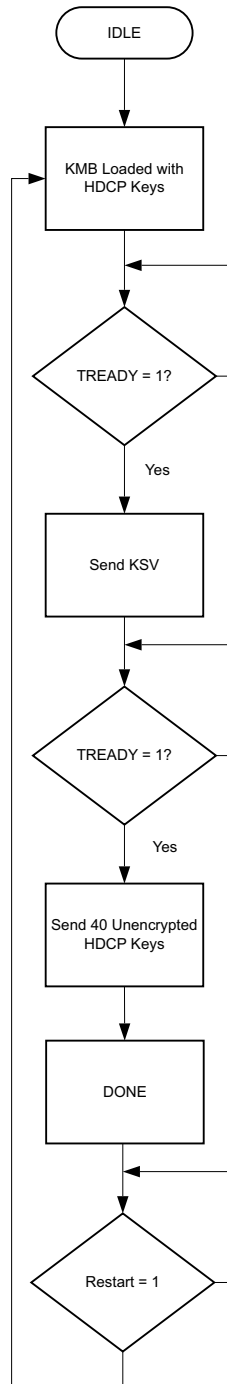
Send Feedback

```
                    IDLE

                         Set Read KSV by writing to Bit[0] of 0x2C        By this time, KMB must be ready with
                                                                          valid KSV

        TREADY to KMB = 1
        (Read KSV from KM
             Block)

                         Local KSV available = 1 (Bit[0] of register 0x30)


        Begin Km cal = 1?
         (Bit[1] of 0x2C)

                                     By this time, KMB must be ready with
                                     valid HDCP Keys

        TREADY to KMB = 1


        Fetch Keys and
        Perform Km
        Calculation

                         Km available = 1 (Bit[1] of register 0x30)


        Km Calculation Done
```

Figure 3-3: **HDCP IP Flow Chart – KMB Interaction**

Send Feedback

Key Management Block Functional Requirements

The main function of the HDCP KMB block is to provide keys to the HDCP IP. Figure 3-4 shows a flowchart and specifies the functional requirements of a KMB.


```
                    ┌──────────┐
                    │   IDLE   │
                    └──────────┘

                  ┌──────────────┐
                  │ KMB Loaded with │
                  │  HDCP Keys   │
                  └──────────────┘

                    TREADY = 1?
                       │ Yes

                  ┌──────────────┐
                  │   Send KSV   │
                  └──────────────┘

                    TREADY = 1?
                       │ Yes

                  ┌──────────────────┐
                  │ Send 40 Unencrypted │
                  │    HDCP Keys      │
                  └──────────────────┘

                  ┌──────────────┐
                  │     DONE     │
                  └──────────────┘

                    Restart = 1
```

X14922-061719

Figure 3-4: **HDCP KMB Expected Behavior**

The notes list out the expected behavior of HDCP KMB.

1. Support AXI4-Stream master interface, as specified by the AXI4-Stream standard. See Figure 3-5.
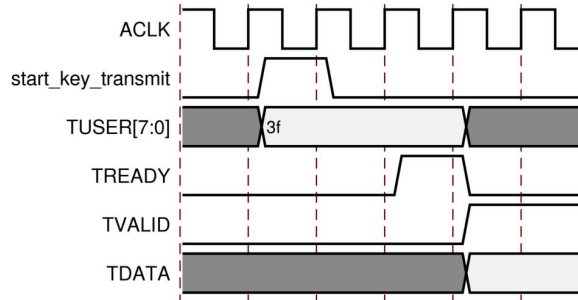
Figure 3-5: **HDCP KMB AXI4-Stream Timing Diagram**

2. The KMB must have two inputs other than the standard AXI4-Stream interface.

   a. `start_reg_key_transmit` – A pulse on this signal indicates that the KMB should restart sending KSV and 40 keys.

   b. `reg_key_select` – This provides an option to store eight sets of HDCP Keys and load one of them based on the value of `reg_key_select`. For example, if `reg_key_select` = 0, load 0th set, and so on. The value of this can be set through the Key Management Control (Register 0x2C) Bits[18:16].

See the HDCP Reference Design for guidance on functional aspects of HDCP Key Management Block.

Security Requirements for Key Management Block

Digital Content Protection. LLC offers restrictions on distribution of HDCP Keys widely. Thus, ensure to design logic so that the keys are maintained securely.

The security of the HDCP Keys is handled in your KMB. Xilinx offers several secure solutions to support this. This document specifies a couple of approaches that can be adopted to design a secure HDCP KMB.

Option 2 offers manufacturing flexibility compared to option 1, as there are standard manufacturing procedures to download data to the EEPROM.

**Option 1**: Store HDCP encrypted keys in the FPGA as a parameter.
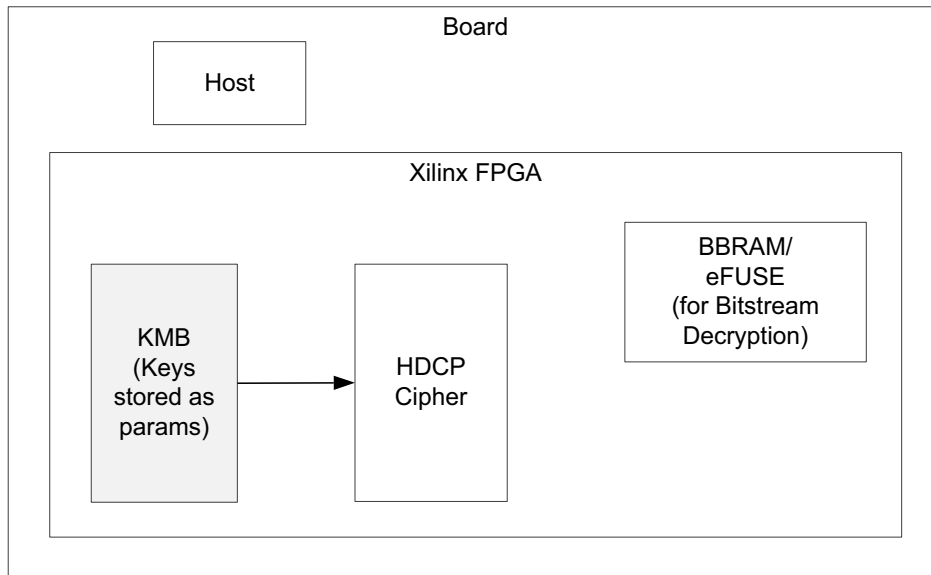
X15035-061719

Figure 3-6: **HDCP KMB Option 1**

- FPGA keys are safe-guarded using bitstream encryption. For more information on the bitstream encryption, BBRAM/eFUSE, and supported devices, see the Xilinx Secure Solutions Page [Ref 7].

- Ensure safety of the HDCP keys by not providing any read access over the AXI4-Lite or any other bus interface.

Option 2: Use EEPROM to store HDCP encrypted keys.
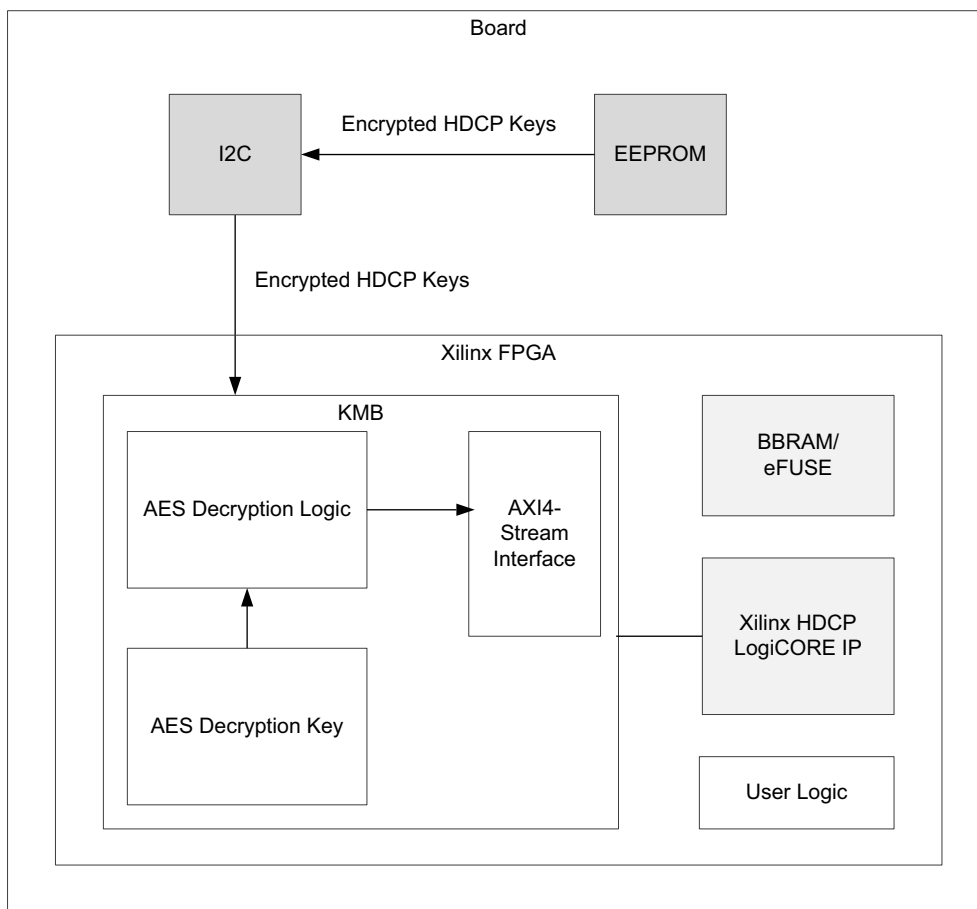
Figure 3-7: **HDCP KMB Option 2**

- Encrypted HDCP Keys are stored in external EEPROM.

- FPGA logic must perform read of these keys over a supported interface like I2C.

- KMB logic decrypts these keys.

- AES key required for the decryption is in the FPGA logic (loose flops). This key is safe-guarded using bitstream encryption. For more information on the bitstream encryption, BBRAM/eFUSE and supported devices, see the Xilinx Secure Solutions Page [Ref 7].

- Ensure safety of the AES key and the decrypted HDCP keys by not providing any read access to those over AXI4-Lite or any other bus interface.

For more information, see the HDCP Key Utility section in *HDMI TX Subsystem Product Guide (PG235)* [Ref 3].

Send Feedback

Clocking

The core uses two clock domains:

- **hdcp_clk** – Most of the core operates in this domain. This clock is driven from the HDMI/DisplayPort controller link_clk domain.

- **s_axi_aclk** – This is the processor domain. It has been tested to run as fast as 200 MHz for DisplayPort.

Resets

The **s_axi_aresetn** is an Active-Low and synchronous to **s_axi_aclk**.

The **hdcp_rst** is a link reset and synchronous to **hdcp_clk**. It is driven from HDMI/DisplayPort controller.

Design Flow Steps

This chapter 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) [Ref 8]

* *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 9]

* *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 10]

* *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 11]

Customizing and Generating the Core

When using HDCP 1.3 or 1.4, see the DisplayPort/HDMI™ reference design respectively.

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 IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 8] 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 Vivado IP catalog. HDCP IP can be generated only as a part of HDMI or DisplayPort subsystem. For example, to use HDCP Encryption with DisplayPort, select **DisplayPort TX** subsystem.

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) [Ref 9] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 10].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

Configuration Tab

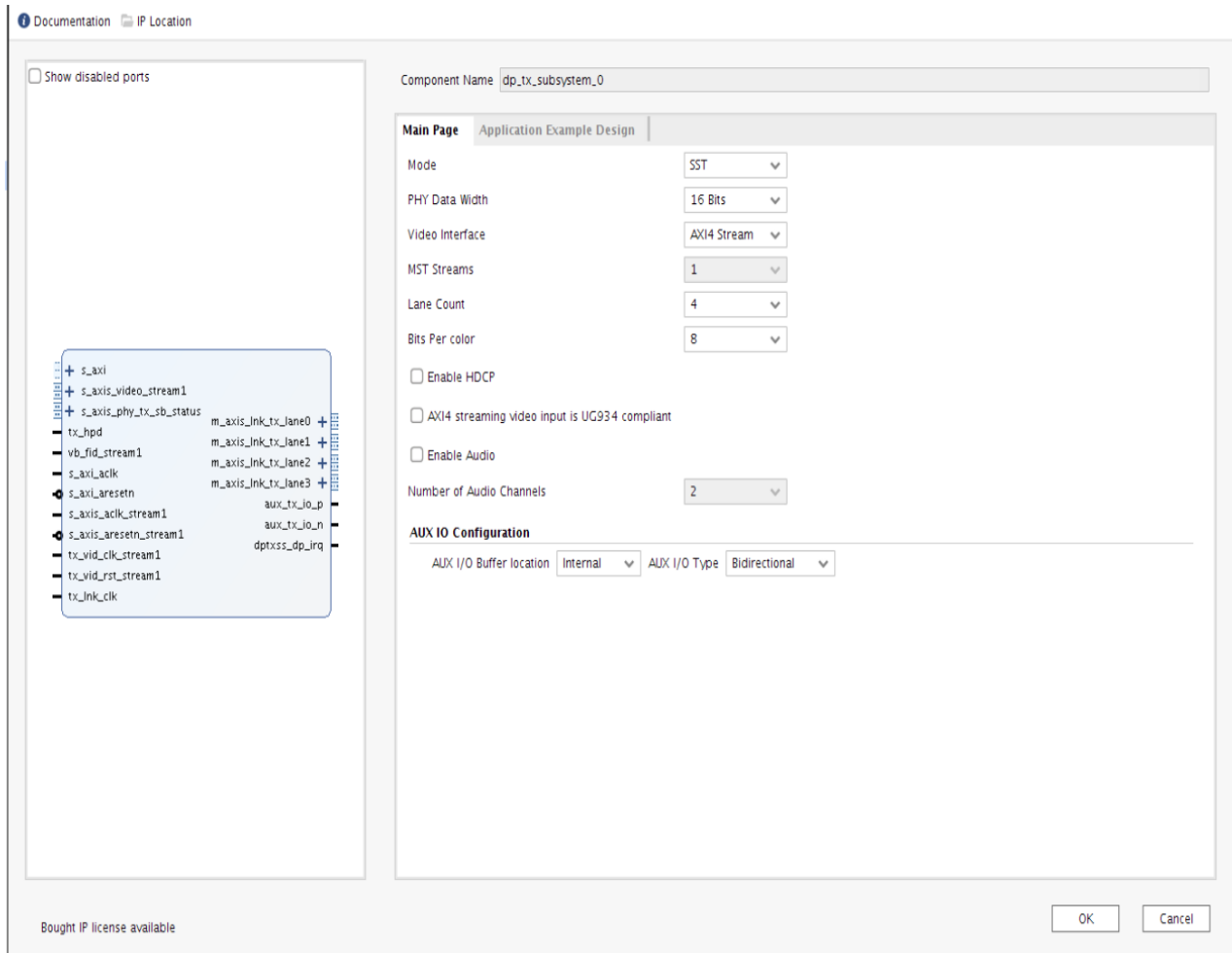Figure 4-1 shows the DisplayPort TX subsystem Vivado IDE main configuration screen.

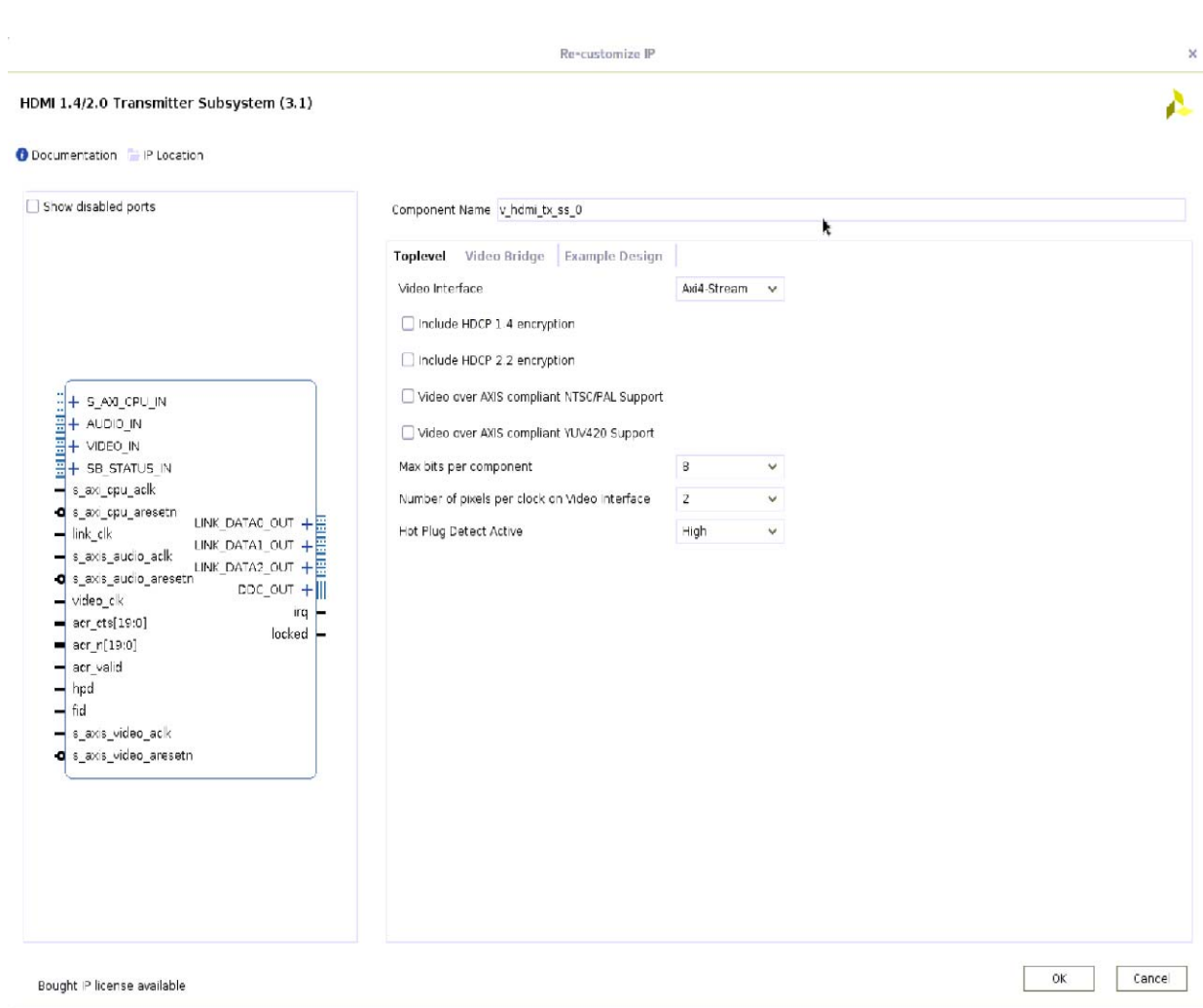*Figure 4-1:* **DisplayPort Transmit Subsystem Configuration Tab**

Figure 4-2 shows the DisplayPort RX subsystem Vivado IDE main configuration screen.
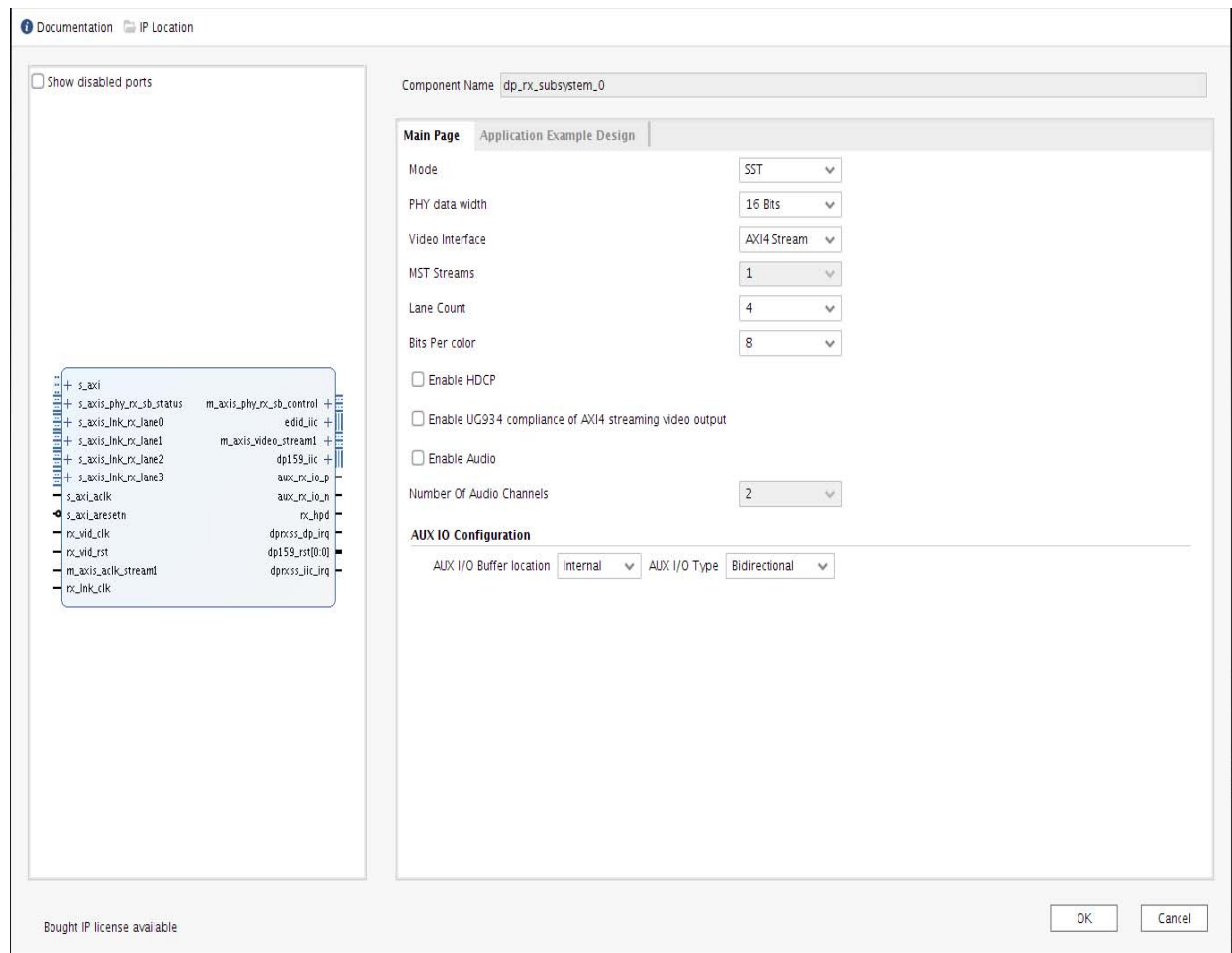
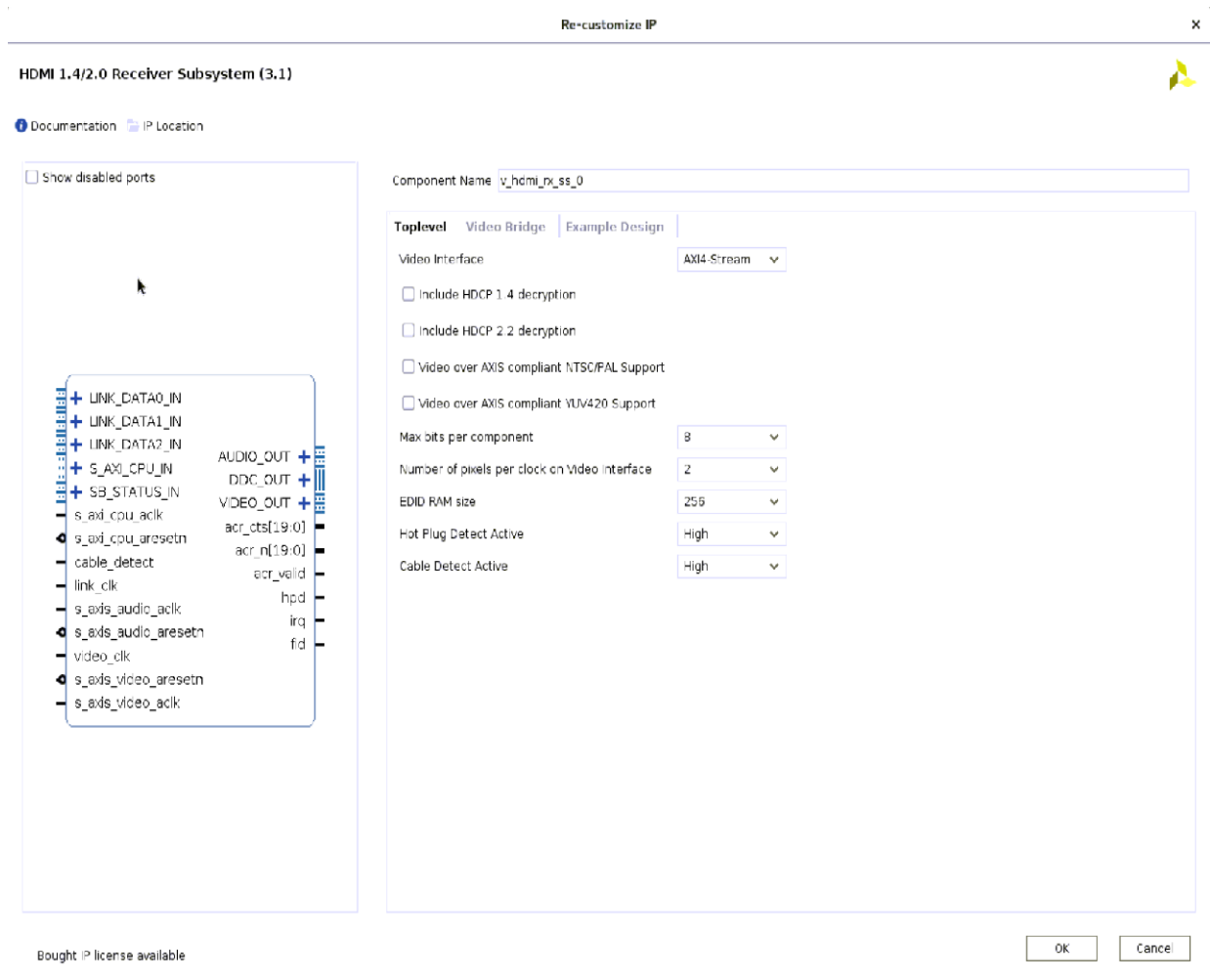*Figure 4-2:* **DisplayPort Receive Subsystem Configuration Tab**

- **Component Name** – The Component Name is used as the name of the top-level wrapper file for the core. The underlying netlist still retains its original name. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9, and "_". The name displayport_0 is used as internal module name and should not be used for the component name.

- **Enable HDCP** – Select to enable HDCP 1.3 Encryption or Decryption for Transmit or Receive subsystems, respectively. Note that HDCP is supported only when DisplayPort is in SST Mode.

Figure 4-3 shows the HDMI TX subsystem Vivado IDE main configuration screen.

Figure 4-3: **HDMI Transmit Subsystem Configuration Tab**

Figure 4-4 shows the HDMI RX subsystem Vivado IDE main configuration screen.

Figure 4-4: **HDMI Receive Subsystem Configuration Tab**

- **Component Name** – The Component Name is used as the name of the top-level wrapper file for the core. The underlying netlist still retains its original name. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9, and "_". The name displayport_0 is used as internal module name and should not be used for the component name.

- **Include HDCP 1.4 encryption** – Select to enable HDCP encryption (for HDMI TX subsystem)

- **Include HDCP 1.4 decryption** – Select to enable HDCP decryption (for HDMI RX subsystem)

Send Feedback

IP Generation

Table 4-1 contains the subsystem information used for HDCP IP generation.

Table 4-1: **HDCP IP Generation**

Mode	Parent IP	Description
DP HDCP – Transmit	DisplayPort TX Subsystem	Use DP TX SS IP. Enable HDCP.
DP HDCP – Receive	DisplayPort RX Subsystem	Use DP RX SS IP. Enable HDCP.
HDMI HDCP – Transmit	HDMI TX Subsystem	Use HDMI TX SS IP. Enable HDCP.
HDMI HDCP – Receive	HDMI RX Subsystem	Use HDMI RX SS IP. Enable HDCP.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 9].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

Add the following constraints in the top-level XDC.

For HDCP 1.4 RX design (HDMI RX design):

```
set_clock_groups -name hdcp_rx_async_group0 -asynchronous -group [get_clocks
-of_objects [get_pins -hier *hdmi_rx_hdcp_cipher_4b_inst/s_axi_aclk]] -group
[get_clocks -of_objects [get_pins -hier *hdmi_rx_hdcp_cipher_4b_inst/hdcp_clk]]
```

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

The clock frequencies are set by the resolution. For more information, see the Maximum Frequencies in Chapter 2.

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

Simulation is not supported for this IP core.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 9].

Verification, Compliance, and Interoperability

This appendix provides details about how this IP core was tested for compliance with the protocol to which it was designed.

The HDCP cores have been verified with extensive hardware testing for HDCP 1.3 DisplayPort and HDCP 1.4 for HDMI™.

Simulation

Simulation is not supported by HDCP cores in HDMI or Display subsystem IP.

Hardware Testing

The HDCP cores in HDMI 1.4/2.0 subsystem IP have been validated on KC705, ZC705, KCU105, VCU118, ZCU102, ZCU104, and ZCU106. Interoperability has been tested.

For DisplayPort, the HDCP core has been validated on KC705 and ZCU102.

Send Feedback

Migrating and Upgrading

This appendix contains information about upgrading to a more recent version of the IP core.

Upgrading in the Vivado Design Suite

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

Debugging

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

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the HDCP, 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.

Documentation

This product guide is the main document associated with the HDCP. 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.

Send Feedback

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 proper 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 at the Xilinx Support web page 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 contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are many tools available to address HDCP 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 with the logic debug 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) [Ref 12].

Send Feedback

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

General Checks

- Always check that the HDCP state machine is not in "Physical-Layer-Down" state. In this case, the state machine does not process the authenticate command in this state. Ensure that the SM comes out this state by posting "PHY-Up" event to the state machine.

- In the case of HDCP transmitter, ensure the system has a timer, preferably dedicated, that counts down 100 ms and post a "timeout" event to the HDCP TX state machine. Without the timer, it is likely that the HDCP TX state machine might never come out of the "Determine-RX" state.

- It is recommended that the HDCP encrypted content fails if you try to play them on a non-HDCP monitor.

Transmitter with DisplayPort Checks

If the Display fails on the sink:

- Ensure that the sink is HDCP capable by reading the HDCP capability in the BCAPS register of the sink.

If the Transmitter reads an invalid BKSV:

- Ensure that the DisplayPort AUX Line is up and the display is coming on the sink before you attempt to read the BKSV.

If the Transmitter reads HDCP incapable from the monitor HHDCP BCAPS register and fails in the "Determine-RX-Capable" state:

- Ensure that the DisplayPort AUX Line is up and the display is coming on the sink before you attempt to read the HDCP BCAPS register.

If the onscreen display fails on encrypting HDCP TX:

- Check if the HDCP is set for the same lane count as the DisplayPort.

- Check if the HDCP keys being used are correct and are being properly loaded in the key management system of the design.

Receiver with DisplayPort Checks

If the HDMI™ Receiver fails to write the BKSV properly and the authentication attempt fails:

* Ensure that the BKSV is properly loaded in the cipher, before reading it and writing it into the HDCP Bksv register.

* Also, check the DisplayPort AUX line is up and working before attempting to write the BKSV to the HDCP DPCD space.

If the authentication fails:

* Ensure that the HDCP receiver computes and writes the Ro within 100 ms. See that there is no additional overheads to delay this.

* Also, make sure the authentication event is posted as soon as the AKSV and Random numbers are written by the transmitter in the HDCP DPCD register space. This is to confirm that the Ro is calculated within 100 ms.

Transmitter with HDMI Checks

If the Display fails on the sink:

* Ensure that the sink is HDCP capable by reading the HDCP capability in the BCAPS register of the sink.

If the Transmitter reads an invalid BKSV:

* Check if the HDMI DDC line is up and the display is coming on the sink before you attempt to read the BKSV.

If the Transmitter reads HDCP incapable from the monitor HHDCP BCAPS register and fails in the "Determine-RX-Capable" state:

* Ensure that the HDMI DDC line is up and the display is coming on the sink before you attempt to read the HDCP BCAPS register.

If HDCP authentication fails:

* Ensure that the interrupt from the cipher for the Ri/Ri' comparisons is properly set. Check if there is no lag in the application because of the additional prints or an unnecessary delay.

* In case the Ro/Ro' fails, try and authenticate again. If this failure persists, check your HDCP keys.

If the onscreen display fails on encrypting HDCP TX:

* Check if the HDCP keys being used are correct and are properly loaded in the key management system of the design.

HDCP 1.x v1.0
PG224 December 20, 2019   www.xilinx.com   **37**
Send Feedback

Receiver with HDMI Checks

If the HDMI Receiver fails to write the BKSV properly and the authentication attempt fails:

- Ensure that the BKSV is properly loaded in the cipher, before reading it and writing it into the HDCP Bksv register.

- Also, make sure the HDMI DDC line is up and working before attempting to write the BKSV to the HDCP DDC space.

If the authentication fails:

- Ensure that the HDCP receiver computes and writes the Ro within 100 ms. Check if there is no additional overheads to delay this.

- Make sure that the authentication event is posted as soon as the AKSV and Random numbers are written by the transmitter in the HDCP DDC register space. This is to confirm that the Ro is calculated within 100 ms.

Application Software Development

This appendix provides information about the HDCP driver and also provides the recommended sequence for authentication in HDCP Receive and HDCP Transmit cores.

HDCP Driver Introduction

The Xilinx® HDCP driver software is provided to allow for the integration of HDCP functionality into designs that make use of the DisplayPort™ and HDMI™ LogiCORE™ IP video interface cores. This software is distributed within the Xilinx SDK (Software Development Kit) and is tightly coupled with the SDK drivers for these interfaces.

This appendix serves as a technical reference describing the architecture and functionality of HDCP driver software as well as providing guidelines for integrating it into a custom DisplayPort and/or HDMI system designs based on Xilinx programmable devices.

Platform Requirements

The current platform requirements for the HDCP driver software are as follows:

- One or more of the following video interfaces (subject to any and all identified Restrictions and Limitations):
  - DisplayPort receive
  - DisplayPort transmit
  - HDMI receive
  - HDMI transmit
- One HDCP 1.x cipher LogiCORE IP instance per video interface that supports HDCP 1.x.
- One HDCP device key set per video interface that supports HDCP 1.x and the associated RTL/IP to interface these to the HDCP 1.x ciphers.
- One timer resource per HDCP 1.x transmit interface that can support durations from a few milliseconds up to several seconds in length.
- Approximately 128 KB of free code space (HDCP specifics only).
- Approximately 1 KB of free data space per HDCP interface (HDCP specifics only).

Restrictions and Limitations

The HDCP driver software has the following restrictions and limitations:

•   DisplayPort receiver and HDMI receiver interfaces are not supported within the same platform design.

DisplayPort transmitter and HDMI transmitter interfaces are not supported within the same platform design.

HDCP Driver Architecture

The architecture of the HDCP software driver can be divided into three distinct submodules.

•   Cipher device

•   Port device

•   Authentication state machine(s)

Each of these modules represents a specific function and/or entity within the driver and work in unison to implement the HDCP software support within the system design. The design of the driver encapsulates all of these submodules into a unified API that is used by the client software application with a single instance of an HDCP interface.

Cipher Device

The cipher device is contained within the HDCP driver software and represents an instance of the HDCP cipher LogiCORE IP block within a system design. Given that the variants of this cipher core provide the same interface to the software, a single implementation of this submodule is used to drive all of them. In the few places where behaviors differ slightly between variants, this software adjusts at runtime based on the contents of a `Type/ID` register within the IP core.

The principal function of the cipher device submodule is to initialize, configure, and monitor the physical encryption cipher used within the HDCP application. As part of this, it provides an interface that allows the software to trigger the reading of the appropriate device key set from the custom key management solution that is attached to the cipher core.

The cipher device submodule is used by higher-level authentication state machines to run re-authentication requests, generate pseudo-random numbers, and enable/disable encryption on individual video streams passing through the cipher core.

It also provides a status output from the actual encryption process that can be used by the authentication state machines to monitor the correct operation of the HDCP function.

The interrupts generated by the cipher core are fully consumed within the HDCP software driver; the client software application needs only to register the cipher core interrupt with the associated vector within the appropriate interrupt controller of the system design. These interrupts are forwarded to the cipher submodule through an appropriate external API of the software driver (**XHdcp1x_CipherIntrHandler**).

The software driver source files that are associated with this submodule are as follows:

- **xhdcp1x_cipher.c**

- **xhdcp1x_cipher.h**

- **xhdcp1x_cipher_intr.c**

- **xhdcp1x_hw.h**

Port Device

The port device(s) that exists within the HDCP software driver represent the management and access to the HDCP register sets that are defined within the HDCP specifications [Ref 14] and [Ref 15]. While the register sets are similar between HDMI and DisplayPort, they do differ, and as a result there are slightly different implementations of these port devices between the two. Regardless of the physical interface type, the transmit port device is always the master of the register set transactions and the receive port device is always the slave.

For transmit port devices, they provide the ability to read and write the HDCP register set that exists on their attached receive (slave) device. These transactions are always run to completion and are initiated solely by the corresponding transmit state machine and/or as part of a debug process if explicitly requested.

For receive port devices, they only provide access to the local HDCP register set; they cannot be used to read the register set from the attached transmit device. Local reads and writes of this register set are performed by the appropriate receive state machine (or debug process) and remote reads and writes can be performed by the transmit port (master) device.

Remote access (either read or write) to a subset of the register set generates notifications (for example, interrupts) to the device that are then in turn posted to the corresponding receive state machine for driving authentication and related procedures. The underlying implementation of the supported HDCP port device modules is completely dependent on the Xilinx BSP device driver for the corresponding video interface type. As a result, the client application software is required to provide the device handle of the video interface that the specific HDCP driver instance is bound to during initialization of the HDCP software driver.

For more information on the BSP Standalone driver, see the following documents:

- *Xilinx Software Developer Kit Help* (UG782) [Ref 16]

Send Feedback

- *OS and Libraries Document Collection* (UG643) [Ref 17]

The internals of the port device rely on the use of an *adapter structure* (a jump table) which maps the HDCP port device API into a protocol (DisplayPort/HDMI) and direction (RX/TX) specific function that implements the associated functionality.

This adapter structure (**XHdcp1x_PortPhyIfAdaptor**) allows for the correct function to be determined and called at run-time based on the exact type of the HDCP interface in question. In some instances (specifically DisplayPort TX), there is no specific port device interrupt defined and, as such, the relevant interrupt(s) (HPD) must be forwarded to the port device submodule through an appropriate external API of the software driver (**XHdcp1x_PortIntrHandler**).

The software driver source files that are associated with this submodule are as follows:

- **xhdcp1x_port.c**
- **xhdcp1x_port.h**
- **xhdcp1x_port_dp.h**
- **xhdcp1x_port_dp_rx.c**
- **xhdcp1x_port_dp_tx.c**
- **xhdcp1x_port_hdmi.h**
- **xhdcp1x_port_hdmi_rx.c**
- **xhdcp1x_port_hdmi_tx.c**
- **xhdcp1x_port_intr.c**

Authentication State Machines

The transmit and receive authentication state machines contained within the driver software mirror those that are defined within the appropriate HDCP specifications [Ref 14] and [Ref 15]. To facilitate integration and debugging, the names of the individual states are aligned with those defined within the specifications wherever possible.

The state machines are responsible for driving the authentication, encryption, and link monitoring processes as requested by the client application software.

The state machines co-ordinate the operation of the cipher, port device(s), and the physical video interface (HDMI or DisplayPort), and rely upon the proper programming and availability of HDCP device keys.

The state machines make use of a simple, OS-independent scheduler to allow for messages and requests to be posted to and from them. This requires the client application software to regularly call a poll function on each HDCP interface to ensure the timely servicing of these messages/requests.

Send Feedback

The transmit state machine provides interfaces that allow for the client application software to request authentication, enable/disable the encryption of video streams, and monitor the progress/outcome of authentication and the ongoing integrity of the encrypted video data. The transmit state must be notified of changes in the physical state of the interface it is running over (either HDMI or DisplayPort) to ensure that any required re-authentication is performed without explicit intervention of the client software application. It fully supports downstream HDCP receivers and repeaters. For a detailed diagram of this state machine, see either figure 2-6 in the *High-bandwidth Digital Content Protection System v1.3 Amendment for DisplayPort* specification [Ref 14] or figure 2-9 in the *High-bandwidth Digital Content Protection System v1.4* specification[Ref 15]. The transmit state definitions are contained within the **XHdcp1x_StateType** contained within the **xhdcp1x_tx.c** source file.

The receive state machine is much simpler than the transmit in that it is not entirely aware of the success/failure of the authentication process. It merely initiates the calculations required for authentication and makes the result available within its HDCP register set for examination by the transmitter. It allows for the client application software to explicitly enable/disable the HDCP capability of the underlying video interface and to query whether encrypted video is being received at any point in time. It also requires that it be notified of changes in the physical interface it is running over so that the appropriate side effects can be initiated. For a detailed diagram of this state machine, see either figure 2-7 in the *High-bandwidth Digital Content Protection System v1.3 Amendment for DisplayPort* specification [Ref 14] or figure 2-10 in the *High-bandwidth Digital Content Protection System v1.4* specification [Ref 15]. The receive state definitions are contained within the **XHdcp1x_StateType** contained within the **xhdcp1x_rx.c** source file.

Both the transmit and receive state machines make use of a link verification procedure to ensure that both ends of the encrypted link remain synchronized over time. This mechanism differs between DisplayPort and HDMI and is detailed in section 2.2.4 of the *High-bandwidth Digital Content Protection System v1.3 Amendment for DisplayPort* specification [Ref 14] and section 2.2.3 of the *High-bandwidth Digital Content Protection System v1.4* specification [Ref 15] respectively.

The software driver source files that are associated with this submodule are as follows:

- **xhdcp1x.c**
- **xhdcp1x.h**
- **xhdcp1x_intr.c**
- **xhdcp1x_platform.c**
- **xhdcp1x_platform.h**
- **xhdcp1x_rx.c**
- **xhdcp1x_rx.h**
- **xhdcp1x_selftest.c**

Send Feedback

- `xhdcp1x_sinit.c`

- `xhdcp1x_tx.c`

- `xhdcp1x_tx.h`

Device Key Management

The device key management software and RTL/IP implementation interact with the HDCP software driver but are, in the strictest sense, outside of the scope of this driver. The Xilinx HDCP solution is designed such that each individual platform integrator is responsible for the security and storage of the HDCP device keys that are issued to them by DCP LLC. The physical interface between the *LogiCORE IP HDCP Cipher Product Guide* and the custom implemented key storage is described in the *High-bandwidth Digital Content Protection System v1.3 Amendment for DisplayPort* specification [Ref 14] and the *High-bandwidth Digital Content Protection System v1.4* specification [Ref 15].

Dependencies

The only dependencies between the HDCP driver software and the actual device key management implementation are, as follows:

- The device keys must be available to be loaded into the HDCP cipher LogiCORE IP prior to initializing the corresponding instance of the HDCP software driver.

- The HDCP software driver supports the programming of *key set selection vector*. This can be used in applications where multiple sets of device keys are available for use. This vector allows for one of up to eight different device key sets to be selected as the data to be loaded when requested by the cipher core. The API function used to set this selection vector is `XHdcp1x_SetKeySelect`.

HDCP Driver Porting Guide

The following system resources are required by the HDCP software driver software:

- Approximately 128 KB of free code space

- Approximately 1 KB of free data space per HDCP driver instance

- Approximately 1 KB of stack space

- One dedicated operating system thread/task for HDCP (if an operating system is being used)

Note: These resource requirements are those needed explicitly for HDCP. The requirements associated with the mandatory video interface(s), HDMI or DisplayPort (DP) are not included.

Integrating Into Your Application

The HDCP software driver can be integrated into any client software application, regardless of whether an operating system is used or not. This integration generally consists of a few simple steps.

1. Allocation of the required number of HDCP driver instances.

2. Binding of the system timer interface functions (only needed if HDCP transmit interfaces are to be used). These functions are used by the state machines to run guard timers where appropriate within the authentication procedure. These functions are, as follows:

    ◦ **XHdcp1x_SetTimerStart**

    ◦ **XHdcp1x_SetTimerStop**

    ◦ **XHdcp1x_SetTimerDelay**

3. Binding of the platform-specific timer expiry handling to call **XHdcp1x_HandleTimeout** when a timer started using the interfaces in step 2 expires.

4. Optional binding of the platform specific function that tests KSV against a stored revocation list (HDCP transmit interfaces only) using **XHdcp1x_SetKsvRevokeCheck**.

5. Initialization, self-test, and basic configuration of the HDCP driver instances. The relevant functions are:

    XHdcp1x_CfgInitialize

    ◦ **XHdcp1x_LookupConfig**

    ◦ **XHdcp1x_SelfTest**

    ◦ **XHdcp1x_SetKeySelect**

6. Binding of the HDCP cipher core interrupt lines and callback functions with the platform interrupt controller(s). The relevant function is **XHdcp1x_CipherIntrHandler**.

7. Registration of the HDCP 1.x port interrupt handler with the corresponding HDMI or DisplayPort device driver. The relevant function is **XHdcp1x_PortIntrHandler**.

    ◦ DisplayPort RX: not used

    ◦ DisplayPort TX: insert into the HPD event and interrupt callback functions

    ◦ HDMI RX/TX: not used

8. Insertion of calls to HDCP *set physical state* and *lane count* function(s) in the appropriate physical interface driver callback functions (HDMI or DisplayPort). The functions of interest are:

    XHdcp1x_SetPhysicalState

    -   DisplayPort RX: insert into the driver callbacks that handle training done (up), training lost (down), bandwidth change (down), and cable unplug (down)

Send Feedback

- DisplayPort TX: insert in the function that starts/trains the DisplayPort link (up or down based on the outcome)

- HDMI RX/TX: call in the stream up/down callback functions

XHdcp1x_SetLaneCount

- DisplayPort RX: insert into the training done driver callback

- DisplayPort TX: insert in the function that starts/trains the DisplayPort link

- HDMI RX/TX: not used

9. Insertion of a call to poll each of the HDCP driver instances within either the main loop of the client software application (bare-metal) or within an HDCP specific thread (operating system). The relevant function within the driver software is **XHdcp1x_Poll.**

10. Insertion of calls where appropriate within the client specific application software to administratively manage and monitor the operation of the HDCP driver instances.

   ○ **XHdcp1x_Authenticate** (transmit interfaces only)

   ○ **XHdcp1x_Disable**

   ○ **XHdcp1x_DisableEncryption** (transmit interfaces only)

   XHdcp1x_Enable

   ○ **XHdcp1x_EnableEncryption** (transmit interfaces only)

   ○ **XHdcp1x_GetEncryption**

   ○ **XHdcp1x_IsAuthenticate**d

   ○ **XHdcp1x_IsInProgress**

It is important to note that the physical video interfaces over which HDCP is to be run must physically exist and be initialized prior to the initialization of the corresponding HDCP driver instance. If this is not the case, then the HDCP driver software will not work as designed. In addition, interrupts on the physical interface should not be enabled until after the corresponding HDCP driver instance has been initialized. While the successful operation of this driver relies on the use interrupts, there are no external interrupt callbacks to be registered specifically by the client software using the HDCP software driver. All of the HDCP-specific interrupts are fully consumed within the software driver and all the client software needs to do is properly complete integration using step 6, step 7, and step 8.

⭐ **IMPORTANT:** *Virtually all of the exported API functions that do not query the state of the driver require a subsequent call to the poll function in order for the request to be fully serviced by the indicated driver instance. As a result, it is crucial that the interface be polled in a timely fashion subsequent to such an API call.*

Send Feedback

Debugging

The HDCP software driver software provides two different mechanisms for facilitating the integration and debug into client application software. These are:

- Runtime debug logging information. You can select the options to enable and disable the details of the HDCP 1.x operations during runtime, respectively.

Figure D-1: **Runtime Debug Logging Information**

- On demand detailed status display. The Information option in the HDCP sub-menu can be used to see the details of HDCP 1.x status

Figure D-2: **On Demand Detailed Status Display**

To enable the runtime debug logging of the driver software, the client application can register a `printf`-style debug logging function with the driver using the appropriate API.

If this is done, then the authentication state machines log all state transitions and periodic status information, providing significant insight into their operation. The function that can be used to register this logging function is `XHdcp1x_SetDebugLogMsg`.

Send Feedback

Note: This debug logging is enabled/disabled on the driver as a whole (as opposed to on a single interface at a time) and does not support priorities of any kind.

In addition to runtime logging, the client application can register a `printf`-style debug print function with the driver software.

This registration permits the driver to perform a one-time dump of detailed status information related to a specific HDCP driver instance when triggered by the client application software. The relevant functions for this are:

- **`XHdcp1x_SetDebugPrintf`**

- **`XHdcp1x_Info`**

The information displayed using the **`XHdcp1x_Info`** function includes details on the current/previous state, encryption status, driver/core version(s), as well as statistics detailing the operation of the authentication state machine (RX and TX) and as its corresponding cipher and port sub-modules. The format of the information displayed differs slightly based on whether the HDCP interface is in the receive or transmit direction.

Receive Authentication

The CE register accesses associated with the receiver authentication should be as follows:

- Write of the Control register to enable the core, enable immediate register updates and set the number of lanes to be used.

- Write the Cipher Control register to set Bit[0] (the XOR function is to be enabled on the receiver for authentication).

- Write of the Key Management Block Control register to load local KSV. Set and then immediately clear Bit[0].

- Read of Key Management Block Status register and check Bit[0] to confirm local KSV available (poll as required).

- Read of the Local KSV (H) and Local KSV (L) registers.

- Write of the Remote KSV (H) and Remote KSV (L) registers with the KSV value sent by the TX.

- Write of the Key Management Block Control register to trigger the calculation of Km. Set and then immediately clear Bit[1]. Set Key Management Block control register to abort Km calculation, and then immediately clear Bit[2].

- Read of the Key Management Status register and check Bit[1] to confirm Km available (poll as required).

- Write of Cipher Bx, By, and Bz registers (with An).

- Read the calculated value of Km and write of Cipher Kx, Ky, and Kz registers (with Km) if needed by the Cipher version.

- Write to Cipher Control register to initiate block cipher. Clear Bit[16] to ensure that the calculated Km value is used. Set and then immediately clear Bit[8].

- Read of Cipher Status register and check Bit[8] to confirm block cipher complete (poll as required).

- Read of Cipher Ri register.

Note: Any step that requires the writing of multiple registers are "book ended" with the clearing and subsequent setting of the register update bit within the Control register.

Transmit Authentication

The CE register accesses associated with the transmitter authentication should be as follows:

- Write of the Control register to enable the core, enable immediate register updates and set the number of lanes to be used.

- Write the Cipher Control register to clear Bit[0] (the XOR function is to be disabled on the transmitter for authentication).

- Write of the Cipher Control register to generate random number. Bit[10] should be set and immediately cleared.

- Read of Cipher Status register and check Bit[10] to confirm RNG complete (poll as required).

- Read of the Cipher Mi (H) and Cipher Mi (L) registers (this value is An).

- Write of the Key Management Control register to load local KSV. Set and then immediately clear Bit[0].

- Read of Key Management Status register and check Bit[0] to confirm local KSV available (poll as required).

- Read of the Local KSV (H) and Local KSV (L) registers.

- Write of the Remote KSV (H) and Remote KSV (L) registers with the value read from the RX.

- Write of the Key Management Control register to calculate Km. Set and immediately clear Bit[1]. Set Key Management Block control register to abort Km calculation, and then immediately clear Bit[2]

- Read of the Key Management Status register and check Bit[1] to confirm Km available (poll as required).

- Write of Cipher Bx, By, and Bz registers (with An + repeater bit).

- Read the calculated value of Km and write of Cipher Kx, Ky, and Kz registers (with Km) if needed by the Cipher version.

- Write to Cipher Control register to initiate block Cipher. Clear Bit[16] to ensure that the calculated Km value is used. Set and then immediately clear Bit[8].

- Read of Cipher Status register and check Bit[8] to confirm block cipher complete (poll as required).

- Read of Cipher Ri register.

Note: Any step that requires the writing of multiple registers are "book ended" with the clearing and subsequent setting of the register update bit within the Control register.

Additional Resources and Legal Notices

Xilinx Resources

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

References

These documents provide supplemental material useful with this product guide:

1. *DisplayPort TX Subsystem Product Guide* (PG199)
2. *DisplayPort RX Subsystem Product Guide* (PG233)
3. *HDMI TX Subsystem Product Guide* (PG235)
4. *HDMI RX Subsystem Product Guide* (PG236)
5. *DisplayPort LogiCORE IP Product Guide* (PG064)
6. *Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs* (XAPP1084)
7. Xilinx Secure Solutions Page
8. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
9. *Vivado Design Suite User Guide: Designing with IP* (UG896)
10. *Vivado Design Suite User Guide: Getting Started* (UG910)
11. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
12. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
13. *Vivado Design Suite User Guide: Implementation* (UG904)
14. *High-bandwidth Digital Content Protection System v1.3* Amendment for Display Port (www.digital-cp.com/hdcp-specifications)
15. *High-bandwidth Digital Content Protection System v1.4* (www.digital-cp.com/hdcp-specifications)

16. *Xilinx Software Developer Kit Help* (UG782)

17. *OS and Libraries Document Collection* (UG643)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/20/2019	1.0	• Added 2 new display ports to Table 2-5. • Replaced instances of R0 with RO.
06/28/2019	1.0	• Updated the resource utilization page. • Updated the license and reference design agreements. • Updated the Repeater Support section for HDCP.
04/26/2016	1.0	Updated cross references
04/06/2016	1.0	• Added driver information. • Updated document title.
12/11/2015	1.0	Updated the license and reference design agreements.
11/18/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices