

WHITEPAPER

AES 256 IP on ***Xilinx* App Store**

AES 256 has a key length of 256 bits, supports the largest bit size, and is practically unbreakable by brute force based on current computing power, making it the strongest encryption standard and more secure.

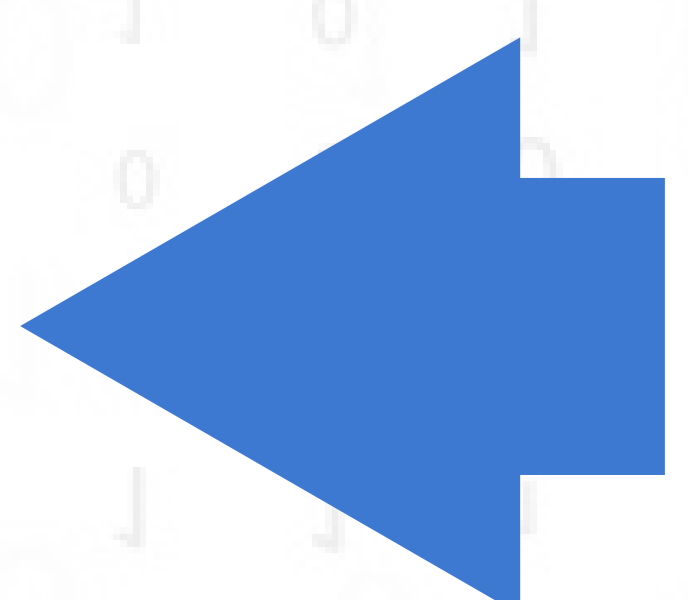
By -

Dheeraj Punia
Hitesh Arora

Contents

Chapter 1: Xilinx App Store	04
What is the Xilinx App Store?.....	04
Features of Xilinx App Store.....	04
3-step Easy Evaluation.....	05
Support for Docker Containers.....	06
Chapter 2: FPGA IP Licensing Principle	06
Licensing Modes.....	07
Licensing Models.....	07
A Protected IP.....	07
DRM Controller IP.....	08
DRM Bus.....	09
DRM Activator IP.....	09
DRM Activator interface with IP to protect.....	10
DRM Hardware integration.....	11
Modify the Design:.....	11
Chapter 3: AES 256 IP	12
Overview.....	12
Block Diagram: AES 256 IP.....	15
Symmetric Encryption vs Asymmetric Encryption.....	14
Key Features & Benefits.....	14
Core Implementation.....	14
FPGA Device Utilization: Post-synthesis results.....	15
Product Release Support: Performance and Quality metrics.....	15
Secure Algorithm: Data-Security and Privacy:.....	16
Chapter 4: Xilinx Vitis Environment	16
Install OpenCL Installable Client Driver Loader.....	16
Vitis Software Platform Installation.....	16
Installing Xilinx Runtime and Data Center Platforms.....	17
Setting Up the Environment to Run the Vitis Software Platform.....	17

Chapter 5: Build and Run App	18
Synthesize the app design.....	18
Compile & Run the Application.....	18
Chapter 6: Docker Containers and App Run	20
Development Environment.....	20
Flow of AES app.....	20
Run the AES 256 App on Xilinx App Store.....	20
Docker Container Details.....	27
Dependency List.....	27
Environment setup.....	27
Development Steps.....	27
Application Usage.....	29
Useful Xilinx commands:.....	29
Chapter 7: Troubleshooting	30
Successful Build.....	30
Use Vitis Analyzer tool to visualize and navigate reports.....	31
[Error] Unable to find DRM controller registers.....	31
[Error] Path is not a valid file: cred.json.....	32
[Error] Metering web service error 400: User account has no entitlement.....	32
[Error] Metering web service error 400.....	33
[XRT] Error: CU was deadlocked? Hardware is not stable.....	34
[Error] Bus Interface property FREQ_HZ does not match between <port_1> and <port_2>.....	35
[XRT] Warning: unaligned host pointer '0x7fffxxxxxx' detected, this lead to extra memcpy.....	35
[XRT] Error: Cannot add a component to the argument.....	36
Check md5sum value of the <file_name>.xclibin.....	36
The first step is you can see which devices are present on your host.....	37
Determine Linux release:.....	37
Unload/reload XRT drivers:.....	38
Flash the card with a deployment platform:.....	38
Reverting the card to factory image:.....	40



Chapter 1: AES 256 IP on Xilinx App Store

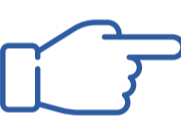
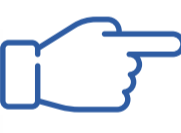
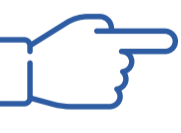

What is the Xilinx App Store?

The Xilinx App Store makes it easy for one to evaluate, purchase, and deploy accelerated applications using cloud-based services or on-premises PC systems as per requirements. It offers a powerful platform to host, market, and sell one's solutions using a managed, easy-to-use, secure Digital Rights Management (DRM) infrastructure. The Xilinx app store is developed by Xilinx and its platform partners and provides pre-built, containerized software that can be easily assessed, bought, and deployed on edge or on the cloud.

The Xilinx App Store provides a powerful platform for hosting, marketing, and distribution of your solutions leveraging a managed, convenient and stable DRM architecture to help multinational consumers accessing their Cloud and/or on-site technologies.

Xilinx app store home page: [Xilinx App Store](#)

Features of Xilinx App Store

-  **Global access and Free trial:** Evaluate for free, sell and purchase globally. The Xilinx App Store provides customers with a one-stop shop for discovering, evaluating, and deploying FPGA-accelerated technologies. Make the ideas with a broad customer base of companies around the world readily discoverable.
-  **Secure and Easy Checkout:** Enabled with payments through Accelize DRM and Stripe. Flexible subscription plans are available. Choose the business-model accounting – subscription, payment for use, time-based or continuous licensing – The App Store DRM infrastructure helps everything that allows you to concentrate on your clients and grow the added value of your solutions. Deliver it as a Docker container or encrypted IP Core, and work with the App Store team to incorporate the IP Digital Rights Management (DRM) into your IP architecture or FPGA app.
-  **Deployment Options:**
 - a. Cloud-based HPC is a partner with Nimbix and Amazon AWS.
 - b. On-premises with Alveo Data accelerator cards/Kria SOMs/PC system.
 - c. Support for Edge and Embedded computing.
-  **Easy & Portable:** Self-service, free trials, online payment, and Containerized apps. Include application specifics and primary advantages in a profile and product catalog listing in the app store. And we're equipped to publish, explore and sell!

3-step Easy Evaluation

1. Select an app as per the requirement.

App Store Container Catalog

Search Filtered Results

Supported Workload ⋮

- Video and Image Processing 9
- Data Analytics 6
- Networking and Security 4
- High Performance Computing 2
- Machine Learning 2

On-Premises Solution ⋮

Results per page 30 60 120 150 Product ▾

Results 1-23 of 23

XILINX High Performance Computing

2D Reverse Time Migration (RTM)

Reverse Time Migration (RTM) is an important seismic imaging

XILINX Data Analytics

Anti-money laundering application

The Anti-money laundering application is used to detect

2. Get Entitlement.

b<>com *Adaptive HDR Converter*

b<>com *Adaptive HDR Converter* offers all the benefits of real-time frame-by-frame adaptive conversion techniques, without any need for manual adjustment. Based on an intelligent algorithm, this technology guarantees an optimal conversion regardless of the video content. Despite the adaptive conversion, it does not require the use of metadata,

Try or Buy

Obtain an entitlement to evaluate or purchase this product.

Evaluate for Free

Begin a free trial and run the application example below.

3. Download and Run.

Results

Once your job is complete, a "case_ali.webp" WebP encoded sample file will be available in **Nimbix FTP**. You can use your internet browser to display it.

You can now use your own input and output folders by using the following command:

```
docker run -v /tmp/cred.json:/opt/ThunderImage-Premium/server/cred.json -v {YourInputFolder
```

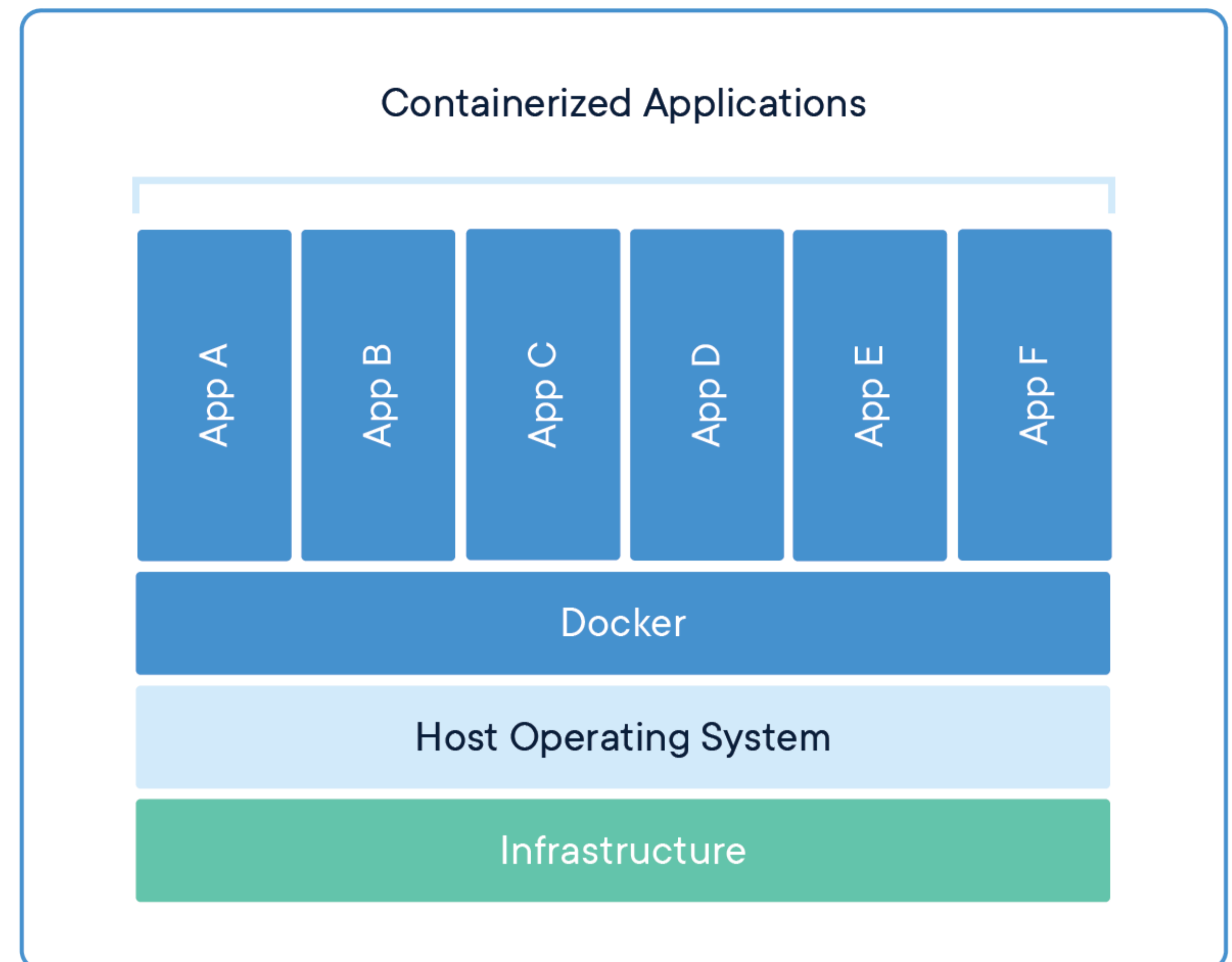


Click to expand

Support for Docker Containers

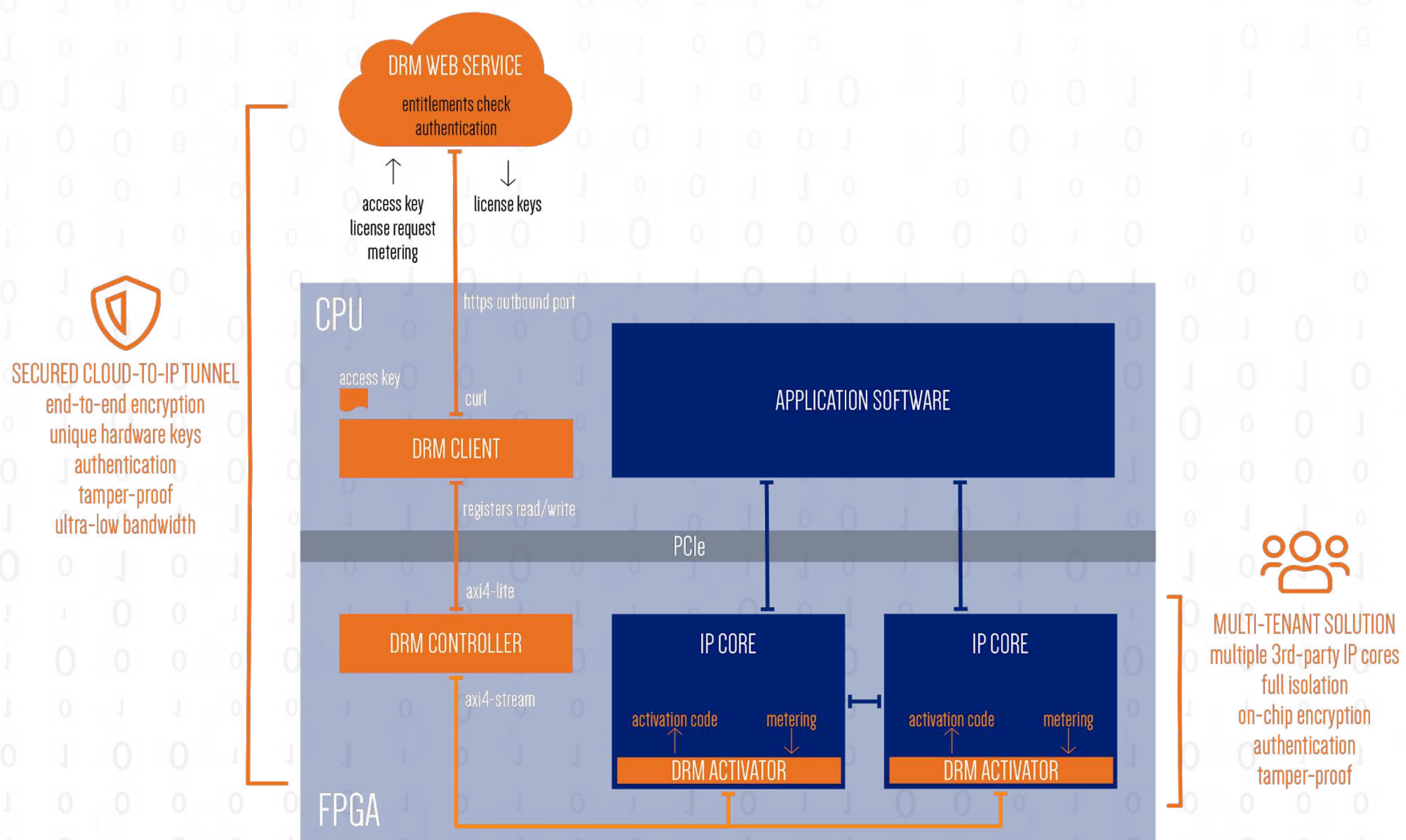
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Separates application dependencies from infrastructure. In loosely separated environments called a container, Docker offers the ability to pack and run the program. The enclosure and safety allow one to operate several containers on a given host at the same time. Containers are lightweight and hold all the necessary applications so that we do not have to focus on what is running on the host at the moment. You can simply swap containers when you're working to make sure anyone who you share has the same workable container.



Chapter 2: FPGA IP Licensing Principle

The core licensing technology is based on a license key that unlocks the proper functioning of an FPGA design at runtime. After its configuration, the license key is loaded into the FPGA, typically through PCIe. The license key is an encrypted container that provides each protected function within the FPGA with a secret Activation Code. This security is enforced in HDL by inserting blocking points into control logic and masking points into datapath logic using individual bits of the activation code. The design is initially locked when the bitstream is inserted into the FPGA board. After that, you'll need the Accelize DRM library to unlock the design with a valid license.



Licensing Modes

- Static Licensing:** A file-based scheme that is deployed by statically packaging the license key into an encrypted license file, locally stored on the FPGA card hosting server.
- Dynamic Licensing:** A server-based scheme that is executed by a license server delivering license keys. Specifically, a standard stream of time-limited single-use license keys is supplied by the license server.

Licensing Models

- Node-Locked Licensing:** This is a static licensing mode. A license grant that requires an application to be executed on, and only on, a particular FPGA card, and is perpetual, transferable, and non-revocable.
- Floating Licensing:** This is a dynamic licensing mode. A Floating License is a license grant that allows execution on any FPGA card of a designated number of concurrent instances of the application.
- Metered Licensing:** This is also a dynamic licensing mode. It facilitates unrestricted deployment of an application on any FPGA card and includes monthly post-use billing based on calculated use. Metered license grants are linked to authenticated users, and the metering information produced within the FPGA is gathered dynamically and securely by the DRM service.

	Nodelocked	Floating	Metered
Licensing mode	Static	Dynamic	Dynamic
entitlement type	fpga card	User	User
Metric	FIXED QUANTITY FPGA cards	FIXED QUANTITY Concurrent nodes Time Data volume	metered usage Concurrent nodes Time Data volume
Validity	Perpetual	Time-based	Time-based
Revocability	×	○	○
Portability	×	○	○
usage monitoring	×	○	○

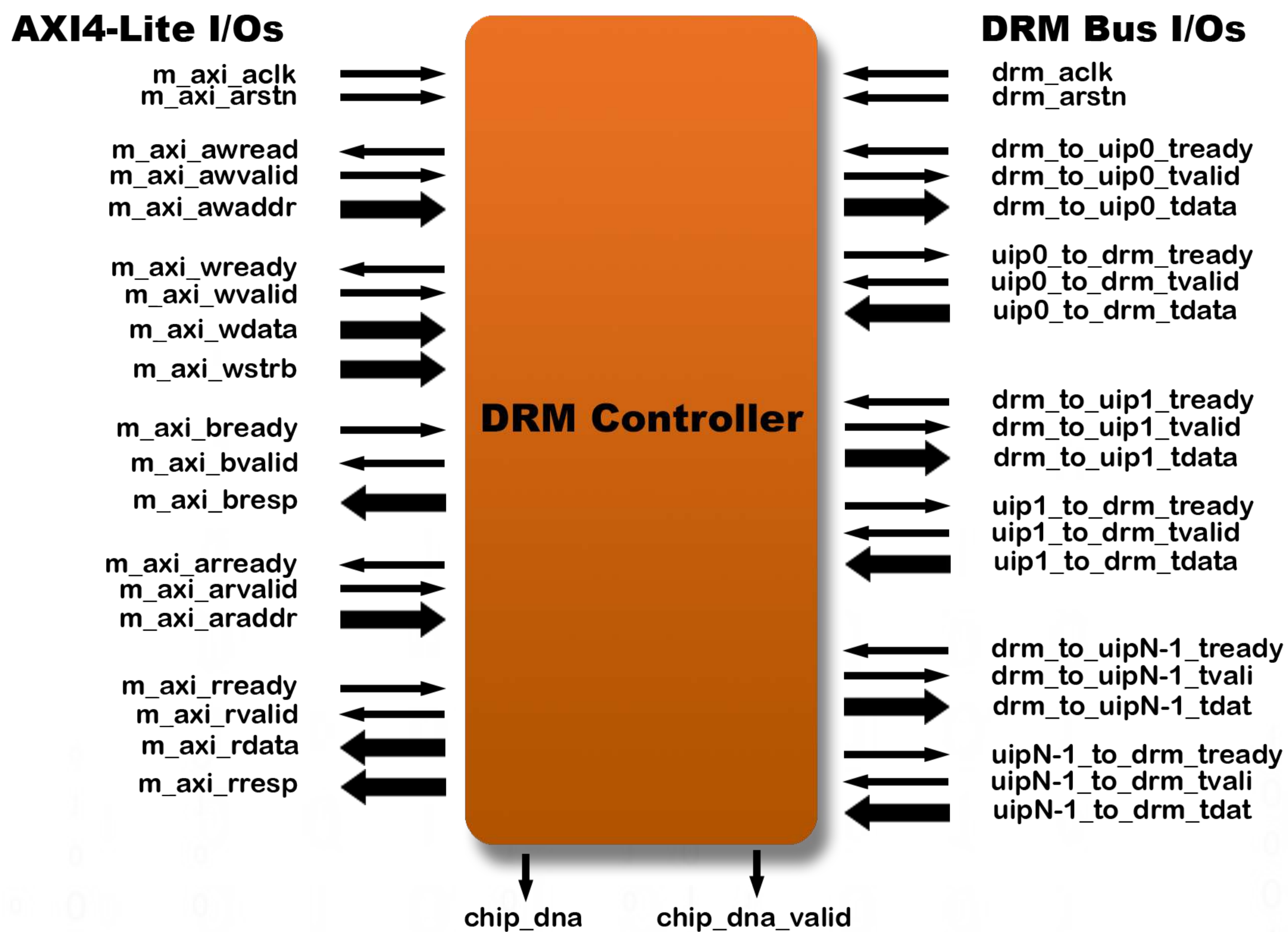
A Protected IP

DRM Controller IP

The DRM Controller controls the transmission of confidential information between the System Software (AXI4-Lite Status & Control interface) and the Protected IP Cores (DRM Bus interface).

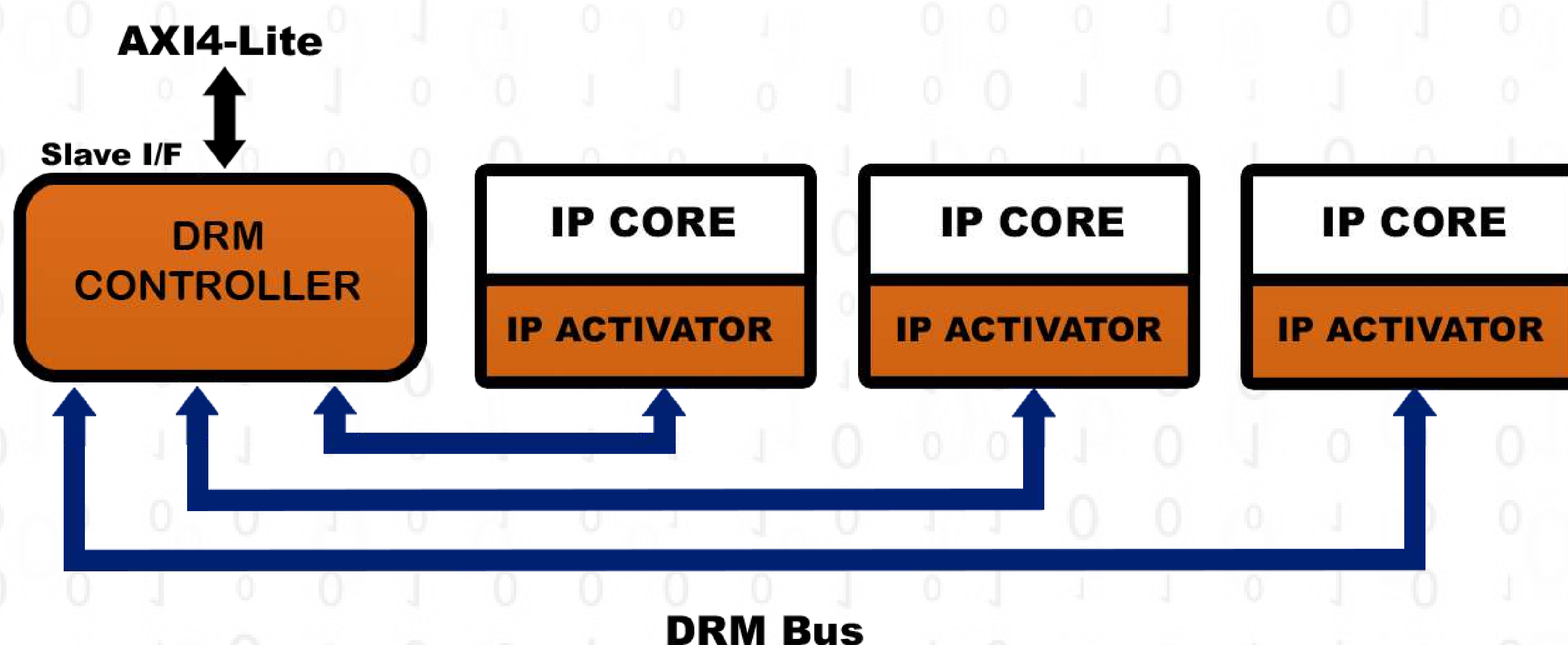
The DRM Controller IP's core functions are as follows:

- ☑ Read and decrypt the encrypted License Key, and send the Activation Codes and Credit timers to the Protected IP Cores in a safe manner.
- ☑ Collect metering data from protected IP cores and send an encrypted and authenticated metering data block to the system.
- ☑ The DRM Controller also collects the design data (Protected IPs VLNVs, 64 bits each) and device identification (Public Chip ID (DNA) or PUF) needed to request the License Key.



In order to service numerous Protected IPs, only one DRM Controller may be created in the Chip Design. Integrate the DRM controller through following the below steps:

- ☑ Give Accelize the amount of protected IP instances you want and they'll provide you the right DRM HDK.
- ☑ At the design's top level, instantiate the DRM Controller.
- ☑ Connect the AXI4-Lite System Bus to the DRM Controller.
- ☑ Activate the DRM Activator and add the DRM interface to protect the IPs.
- ☑ Connect the DRM Controller to the various protected IP instances.



DRM Bus

The AXI4-Stream protocol is used to communicate on the DRM Bus, with the IP Activator acting as a slave and the DRM Controller acting as the master. The number of Protected IP cores in the architecture determines the size of the DRM bus. There are three sections of each IP connection:

The Clock and Reset ports

Name	Direction	Size	Description
drm_aclk	in	1	DRM bus clock: must be identical to all the DRM Activators clock
drm_arstn	in	1	DRM bus asynchronous active low reset

The DRM Controller to Activator channel

Name	Direction	Size	Description
drm_to_uip<IDX>_tready	in	1	AXI4-Stream Ready signal for DRM Controller to IP Activator Channel
drm_to_uip<IDX>_tvalid	in	1	AXI4-Stream Valid signal for DRM Controller to IP Activator Channel
drm_to_uip<IDX>_tdata	in	32	AXI4-Stream Data signal for DRM Controller to IP Activator Channel

The Activator to DRM Controller channel

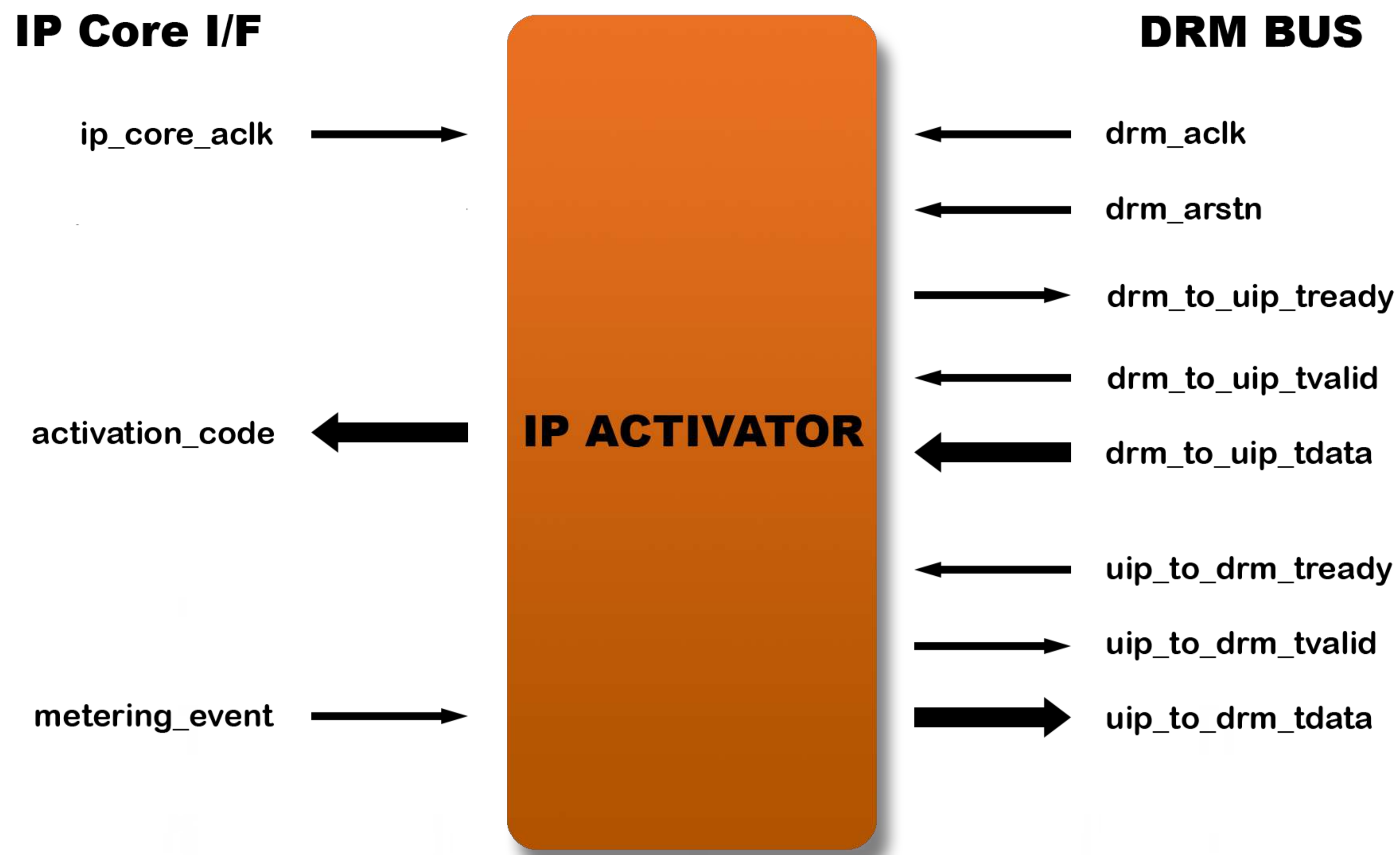
Name	Direction	Size	Description
uip<IDX>_to_drm_tready	out	1	AXI4-Stream Ready signal for IP Activator to DRM Controller Channel
uip<IDX>_to_drm_tvalid	in	1	AXI4-Stream Valid signal for IP Activator to DRM Controller Channel
uip<IDX>_to_drm_tdata	in	32	AXI4-Stream Data signal for IP Activator to DRM Controller Channel

DRM Activator IP

Communication on the DRM Bus uses an AXI4-Stream protocol where the IP Activator is a slave and the DRM Controller is the master. A single Controller is always required but any number of Activators is supported (1 to N connections).

The main functionality of the DRM Activator IP is to:

- ✓ Deliver a 128 bits Activation Code to the IP Core for behavior control.
- ✓ Maintain a credit timer for time based activation.
- ✓ Store a metering counter for activities measurement.



DRM Activator interface with IP to protect

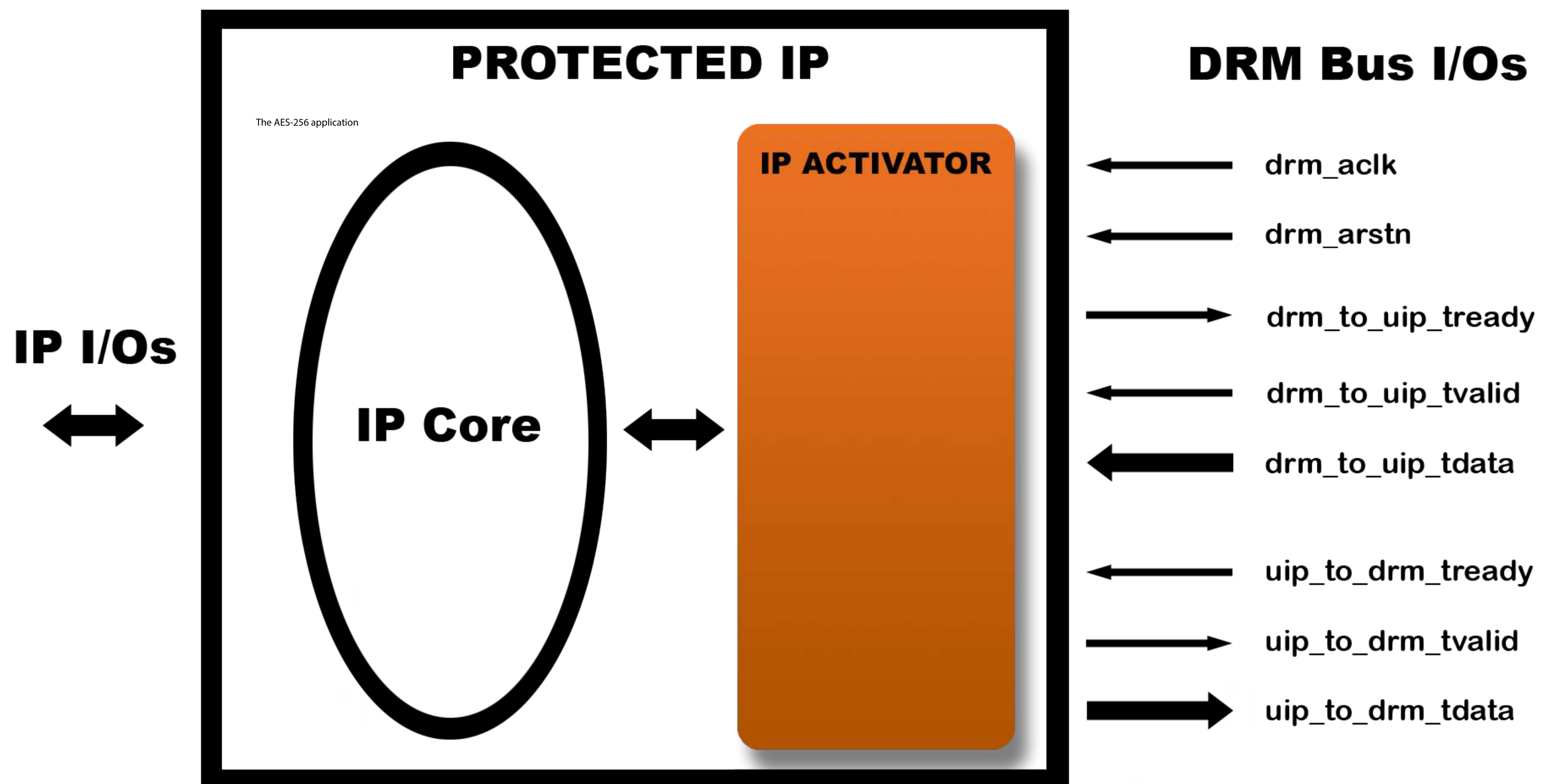
The interface with the IP Core is a simple register interface with control signals.

Name	Direction	Size	Description
ip_core_aclk	in	1	IP Core clock
metering_event	in	1	Level-sensitive signal synchronous to ip_core_aclk that increments the Metering counter when sampled to '1'
activation_code	out	128	Activation Code as provided by the License Key currently loaded.

IP core signals that interact with the DRM Activator must be synced with the IP Core clock domain, ip_core_aclk signal. The IP core will implement its own CDC on an internal level. FSM transition and datapath gates are controlled by the IP Core using the 128 bits of the Activation Code output.

The DRM Activator has an inbuilt 64-bit Metering counter that stores the IP Core's activities. When the session is ended, it is synchronously reset through the DRM Bus protocol. It is increased by asserting the metering_event input for 1 clock cycle under the direction of the IP Core. Because the metering_event is a level-sensitive signal, ensure it is de-asserted once the event is over.

By using conditional logic depending on the Activation Code value, you may protect certain important code (128 bits). Count data metrics associated with IP consumption (bytes, frames, or any other unit) and produce a pulse on the DRM Activator event input for each usage unit.



DRM Hardware integration

You will receive a zip file from Accelize. It has three folders that include the HDK sources:

- ✓ The **common** folder contains the activator and controller's IP common structure.
- ✓ The top-level VHDL controller and the Verilog Wrapper are both found in the **controller** folder. Each IP instance in your design has two AXI4-Stream interfaces, thus the controller has the right amount of ports (already protected IPs and IPs to protect).
- ✓ The **activator** folder contains the VHDL core for the activator as well as numerous simulation and synthesis wrappers. For each IP core type, a single DRM Activator is sent. The same activator will be invoked many times by several instances of the same IP core.

Modify the Design:

- ✓ **Protect the IP cores:** This can be accomplished in a variety of ways. We propose to develop a wrapper in which the DRM Activator and the IP core are instantiated. To integrate DRM protection and usage measurement algorithms, the original IP core must be significantly updated. Managing numerous instances of the same protected IP is built-in with this technique.
- ✓ **Create a wrapper:** The wrapper interface combines the IP interface with the DRM AXI4-Stream interface for communication with the DRM Controller.
- ✓ The most important step is to intelligently change the original IP core such that a piece of IP internal logic is paired with the activation code bits given by the DRM activator signal to activate or deactivate part or all of the IP capabilities.
- ✓ The activation code's 128 bits are utilized to define criteria for IP activation and deactivation. Individual bits, groups of bits, and ranges of bits can be utilized in the IP code to instrument it in various ways such as gate signals, switch FSM states, and select functional parts.

- ✓ In the wrapper, instantiate the customized IP core and DRM Activator and connect them. They may be instantiated once or several times in your FPGA design once your IP is secured.
- ✓ You are good to move ahead. **Now simulate your design.**

Chapter 3: AES 256 IP

Overview

The Rijndael cipher was chosen as the symmetric key ciphering algorithm in the AES specification, which stands for "Advanced Encryption Standard." AES encrypts a message using a private key that can only be decrypted by the key holder. This is useful for a variety of purposes, but one example is a laptop that encrypts the contents of the hard disk when it is idle.

A state is a matrix of bytes that the AES utilizes to operate. The plain text is transformed into the final ciphertext after many rounds of transformation. One round reads the state into four 4-byte variables and transforms them, XOR's them using a 32-byte round key, and stores the results. In compliance with the NIST Advanced Encryption Standard, the AES encryption IP core implements Rijndael encoding and decoding. It works with 256-bit blocks and is programmed to work with 256-bit key length.

The AES 256 algorithm processes plain data blocks of 128 bits, generates cipher data blocks of 128 bits using cipher keys of 256 bits (32 bytes). AES uses symmetric key encryption, which involves the use of only one secret key to cipher and deciphers the information. The AES-256 application performs some encryption algorithms on the data provided by the user in the form of the file. The data is then pushed to the Alveo U200 card in the form of a buffer, the device will perform the encryption on the plaintext and send back the ciphertext in the form of a buffer. After the encryption the user will get an output file with the encrypted data in it. In the AES256 application, we don't have any C/C++ kernel, the encryption happens in the RTL kernel.

AES 256 has a key length of 256 bits, supports the largest bit size, and is practically unbreakable by brute force based on current computing power, making it the strongest encryption standard and more secure.

Key Size	Possible Combinations	Rounds	Time to Crack (years)
128	3.4×10^{38}	10	1.02×10^{18}
192	6.2×10^{57}	12	1.872×10^{37}
256	1.1×10^{77}	14	3.31×10^{56}

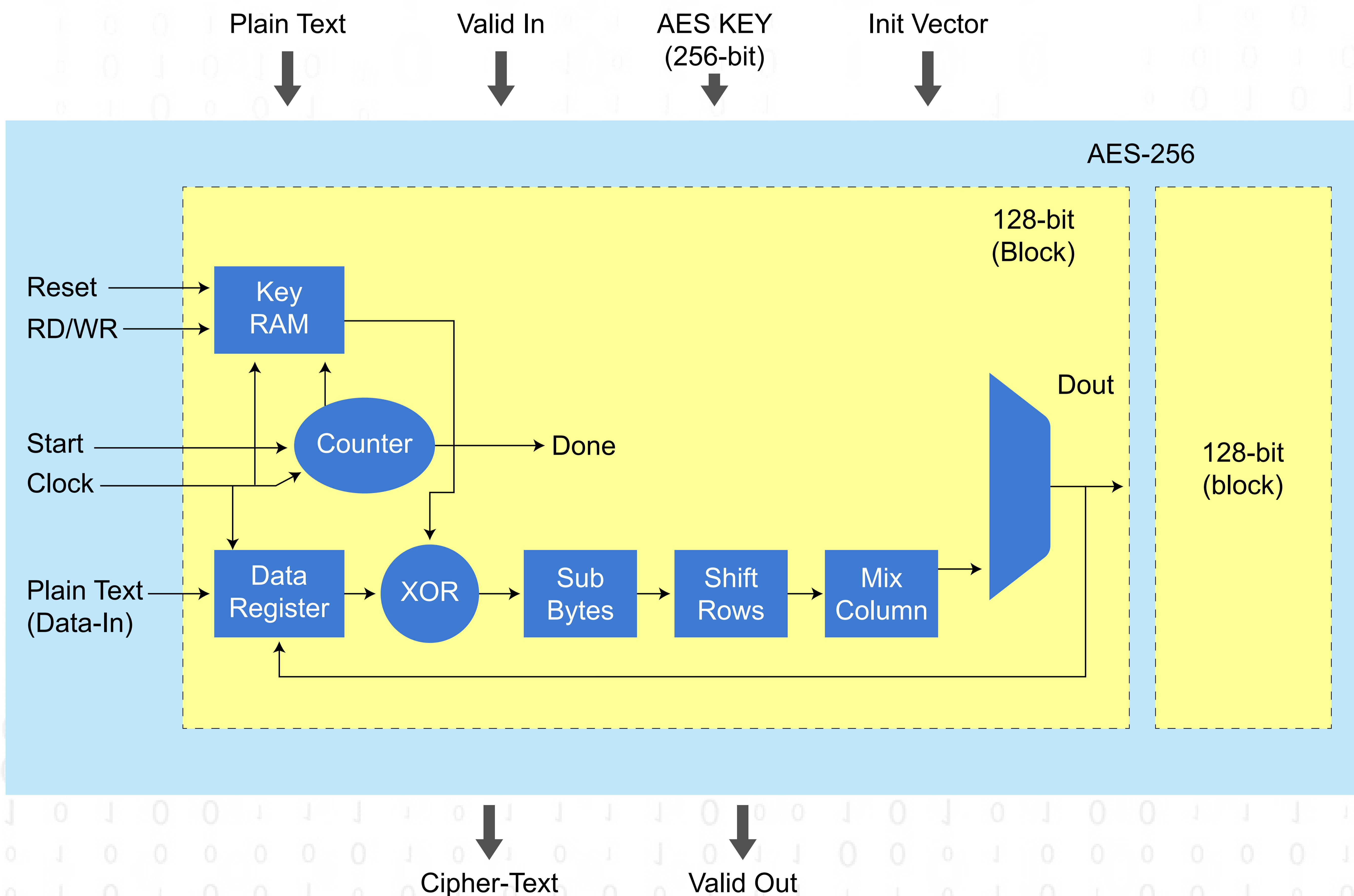
Symmetric Encryption



Asymmetric Encryption



Block Diagram: AES 256 IP



Symmetric Encryption vs Asymmetric Encryption

Attributes	Asymmetric	Symmetric
Keys	One entity has a public key and the other entity has a private key	One key is shared between two or more entities
Key Exchange	Distributed inbound	Out of bound
Speed	The algorithm is more complex and slower	The algorithm is less complex and faster
Number of keys	Grows linearly as users grow	Grows exponentially as users grow
Use	Key encryption and distributing keys	Bulk encryption
Security	Confidentiality, authentication, non-repudiation	Confidentiality

Key Features & Benefits

- ✓ Data Path runs on 100 MHz X 256 width.
- ✓ Programming of Key and Initialization Vector Supported.
- ✓ Buffer-free implementation of RTL code is fast and easy to integrate into SoCs.
- ✓ Pipelined instances architecture with Vendor-independent code.
- ✓ Support for CTR mode. On-demand availability for CBC mode.
- ✓ Solution Type: IP Core.
- ✓ End-Market: Automotive, Broadcast, Consumer, Industrial, Medical, Military, Computer & Storage, Wireless.

Core Implementation

The input signals are synchronized and sampled on the clock's rising side. Flip-flops drive output signals, which are not coupled directly to input signals by combinational circuits.

Signal Name	Signal width	Mode	Description
clk	1	in	Clock signal. Sets up the operational clock frequency.
rst	1	in	Reset signal.
init_vector	256	in	The initialization vector is an arbitrary number that is used along with the secret key for data encryption.

plaintext	256	in	A message before encryption or after decryption. (Input data)
cipherkey	256	in	AES Cipher key is used for enciphering and deciphering data.
valid_in	1	in	This indicates the plain text is valid to process at the input terminal.
valid_out	1	out	This indicates the cipher-text is encrypted correctly and available at the output terminal.
ciphertext	256	out	Encrypted data are available to send or use accordingly.

FPGA Device Utilization: Post-synthesis results

Tool/IDE	Xilinx Vivado
Number of Slice Registers/FF	18511
Number of Slice LUTs	48528
Number of LUTRAM	1792
Number of Block RAM	4

Product Release Support: Performance and Quality metrics

Maximum Frequency	125 MHz
Throughput	It is a pipelined design. (up to 256/8 Gbps)
Input-to-Output Delay	16 clock-cycles

IP Quality Metrics	Generic
Simulators supported	Modelsim, Xilinx Vivado, Lattice Radiant
Hardware validated	Validated on major FPGA Devices

Source language	VHDL/Verilog
Testbench language	VHDL
Software drivers provided	On-demand availabilities

IP Quality Metrics	Deliverables
Design file (encrypted source code/post-synthesis netlist)	Target specific netlist/fully synthesizable source code
Testbench or design example	Simulation model and testbench with FIPS test vectors
Documentation with revision control	Included
Readme file	Yes
Additional customer deliverables	On-demand availabilities

Secure Algorithm: Data-Security and Privacy:

- ✓ Cache Attack: The IP doesn't have any memory elements. It doesn't hold the plain text, keys, and initialization vector in the system.
- ✓ Timing Attack: The time to complete the encryption or decryption doesn't depend on the complexity of the key/initialization vector. So the execution time depends on the size of plain text.
- ✓ Power Monitoring Attack: The power variation of the hardware is very low.
- ✓ EM - Fault Analysis Attacks: Hardware design(board EM shielding) makes sure the secret data is secure.
- ✓ Masking and temporal noise insertion with desynchronization make the IP SPA & DPA secure. (TRNG code block can be introduced to generate a random number to mask the actual transactions - TBD)

Chapter 4: Xilinx Vitis Environment

Note: The steps and procedure given in this section is applicable to PC running on **Ubuntu**.

Install OpenCL Installable Client Driver Loader

On Ubuntu, the ICD library is packaged with the distribution. Install the following packages:

- ✓ ocl-icd-libopencl1
- ✓ opencl-headers
- ✓ ocl-icd-opencl-dev

Vitis Software Platform Installation

1. Go to the Xilinx Downloads Website.
2. Download the installer for your operating system.
3. Run the installer, which opens the Xilinx Unified 2020.2 Installer. Click Next.

4. Enter your Xilinx user account credentials, and then select Download and Install Now. Click Next.
5. Accept the terms and conditions by clicking each I Agree checkbox. Click Next.
6. Select Vitis, and then click Next.
7. Optionally, customize your installation by selecting design tools and devices, and then click Next.
8. Select the installation directory, optional shortcut, and file association options, and then click Next.
9. Review the installation summary, which shows the options and locations you have selected.
10. To proceed with the installation of the Vitis software platform, click Install.

Installing Xilinx Runtime and Data Center Platforms

1. To download the DEB file, go to the Alveo Packages webpage.
2. Select your platform and operating system and proceed and follow the steps 1 to 3 on the webpage or continue as below:

- i. **Download the Xilinx Runtime**

The Xilinx runtime (XRT) is a low-level communication layer (APIs and drivers) between the host and the card. Download the package and enter the command to install the package.

Install using: `sudo apt install <deb-dir>/<xrt_filename_OS>.deb`

- ii. **Installing Data Center Platforms**

Download the Deployment Target Platform: The deployment target platform is the communication layer physically implemented and flashed into the card. Download the package.

Install using: `sudo apt install <deb-dir>/<deployment_shell_filename_OS>.deb`

Download the Development Target Platform: The development target platform is required if you are building your own applications. Download the package.

Install using: `sudo apt install <deb-dir>/<development_shell_filename_OS>.deb`

Setting Up the Environment to Run the Vitis Software Platform

To configure the environment to run the Vitis software platform, run the following scripts, which set up the environment to run in a specific command shell.

```
#setup: XILINX_VITIS and XILINX_VIVADO variables
source <Vitis_install_path>/Vitis/2020.2/settings64.sh
#setup: XILINX_XRT
source /opt/xilinx/xrt/setup.sh
```

After setting up the environment enter the command to open the Vitis IDE: `vitis`
(Required if manually working with the IDE. Often used with command-line tools and scripts.)

To specify the location of any platforms you have installed as directed in Installing Data Center Platforms, set the following environment variable:

```
export PLATFORM_REPO_PATHS=<path to platforms>
```

Chapter 5: Build and Run App

Synthesize the app design

1. Edit Makefile: The one located in the parent folder of the app project. The required variables are set as per the specified directory structure at the original app build time. Please update the required variables as per the directory structure in your project and PC.

- i. Change "VTS_PLATFORM" variable
- ii. Change "OUTPUT_DIR" variable
- iii. Change "KERNEL_FREQ_MHZ" variable [Optional, if required]

2. Setup Vitis and XRT Environment

```
#setup: XILINX_VITIS and XILINX_VIVADO variables
source <Vitis_install_path>/Vitis/2020.2/settings64.sh
#setup: XILINX_XRT
source /opt/xilinx/xrt/setup.sh
```

3. Launch synthesis

- i. Enter command: `make clean_all` (clears previous object and output files)
- ii. Enter command: `make` (launches the app build process)

Compile & Run the Application

Prerequisites: If designing and developing any new application with Accelize DRM.

1. Create an account on [Accelize Portal] (<https://portal.accelize.com>)
2. Create your Access Key on [Accelize Portal - Access Key] (<https://portal.accelize.com/front/customer/apicredential>)
3. Follow and Install [Accelize DRM Library] (http://accelize.s3-website-eu-west-1.amazonaws.com/documentation/stable/drm_library_installation.html#installation-from-packages) version 2.3 or higher

Replace "`app/{your-exec-env}/cred.json`" with your Access Key

Edit "`app/{your-exec-env}/conf.json`" to change "boardType" and "frequency" parameters [Optional]

On-Premise Execution

```
cd app
source /opt/xilinx/xrt/setup.sh
make clean all
./app {path-to-xclbin}
```

Minimum System Requirements

The minimum system requirements for running the Alveo™ U200 Data Center accelerator cards are listed below:

Component	Requirement
Motherboard	PCI Express® 3.0-compatible with one dual-width x16 slot
System Power Supply	225W via PCI Express Slot connection and 8-pin PCI Express Auxiliary Power cable.
Operating System	Linux, 64-bit: <ul style="list-style-type: none"> • Ubuntu 16.04, 18.04 • CentOS 7.4, 7.5, 7.6 • RHEL 7.4, 7.5, 7.6
System Memory	For deployment installations, a minimum of 16 GB plus application memory requirements is required. For development installations, a minimum of 64 GB of device memory is required, but 80 GB is recommended.
Internet Connection	Required for downloading drivers and utilities.
Hard disk space	Satisfy the minimum system requirements for your operating system.

Development Environment

Component	Output (host machine)
OS Version	Operating System: Ubuntu 18.04.5 LTS
Kernel Version	Kernel: Linux 5.4.0-73-generic
Vitis Version	Vitis v2020.2 (64-bit)
Driver Used	Xclmgmt , xocl
Specification about the card	Card type: u200 Flash type: SPI Flashable partition running on FPGA: xilinx_u200_xdma_201830_2, [ID=0x5d1211e8],[SC=4.2.0] Flashable partitions installed in system: xilinx_u200_xdma_201830_2, [ID=0x5d1211e8],[SC=4.2.0]
BSP Version	[0000:01:00.0]
BSP Version	20.10.6

Chapter 6: Docker Containers and App Run

Development Environment

Component	Requirement
Motherboard	PCI Express® 3.0-compatible with one dual-width x16 slot
System Power Supply	225W via PCI Express Slot connection and 8-pin PCI Express Auxiliary Power cable.
Operating System	Linux, 64-bit: • Ubuntu 16.04, 18.04 • CentOS 7.4, 7.5, 7.6 • RHEL 7.4, 7.5, 7.6
System Memory	For deployment installations, a minimum of 16 GB plus application memory requirements is required. For development installations, a minimum of 64 GB of device memory is required, but 80 GB is recommended.
Internet Connection	Required for downloading drivers and utilities.
Hard disk space	Satisfy the minimum system requirements for your operating system.
Platform	Alveo U200

Flow of AES app

1. Open Xilinx Device and Load the XCLBIN.
2. Set up the Buffers that are used to transfer the data between the host and the device.
3. Use the Buffer APIs for the data transfer between host and device (before and after the kernel execution).
4. Use Kernel and Run handle/objects to offload and manage the compute-intensive tasks running on FPGA.

Run the AES 256 App on Xilinx App Store

1. Obtain an Account Access Key: An access key is required to authenticate a user and grant them access to the application based on their entitlements. To obtain your account access key, follow these steps:
 - i. Login to [Xilinx App Store](#)
 - ii. Click the button labeled "**Manage Account**" to view entitlements.
 - iii. Click the "**Access Key**" link on the left side menu
 - iv. Click the "**Create an Access Key**" button.
 - v. Download the resulting file "**cred.json**" to the home location or recommended to in /tmp folder

2. **[Optional]** If you generate the cred.json file you can directly run the application by running the demo script just for dry run.

3. **Host Setup:** The Xilinx Runtime (XRT) host application is supported on Ubuntu 16.04 /18.04 and CentOS 7.x. With sudo access, use the following command to download and run the setup script:

- i. Clone GitHub Repository for Xilinx Base Runtime
`git clone https://github.com/Xilinx/Xilinx_Base_Runtime.git`
- ii. Go to the Xilinx Base Runtime
`cd Xilinx_base_Runtime`
- iii. Run the Host Setup Script
`./host_setup.sh -v 2020.2`

Note: Please wait for the installation to complete. During this time you may need press [Y] to continue the host setup.

If you choose to flash the FPGA, you will need to cold reboot the local machine after the installation is completed to load the new image on the FPGA. The script for host setup can be used to set up other versions XRT and shell. Please check https://github.com/Xilinx/Xilinx_Base_Runtime for more details.

4. **Install Docker (If not installed already):** With sudo access, use the following command to run the utility script to install docker.

- a. Go to Xilinx_Base_Runtime utilities directory
`cd Xilinx_Base_Runtime/utilities`
- b. Run the Docker installation script
`./docker_install.sh`

This command will let us install docker on the host machine.

To verify that the Docker has been properly installed you can run

```
docker run hello-world
```

By default the Docker daemon will check if the host machine has the image of the hello-world if not found it will automatically pull the images from the Docker Hub.

If docker is already installed, enable it:

```
systemctl restart docker  
systemctl enable docker
```

Application Execution

Enter the following commands in a terminal window to run the application:

1. Setup Environment Variables by script from Xilinx_Base_Runtime

```
source Xilinx_Base_Runtime/utilities/xilinx_docker_setup.sh
```

2. Pull the Docker Image from Docker Hub

```
docker pull xilinx/hubxilinx/logicfruit_aes256_u200:latest
```

3. Run this single command to run the Docker Image and the AES_APP

```
docker run --rm -v $(pwd)/cred.json:/cred.json -v bin_test.bin:/AES/bin_test.bin $XILINX_DOCKER_DEVICES --shm-size=64Ghubxilinx/logicfruit_aes256_u200:latest
```

Description of the Command Arguments

- ✓ --rm : Automatically remove the container when it exits.
- ✓ -v /(Absolute path from the host machine)/cred.json:/cred.json - Map local cred.json dir : container ("/") directory.
- ✓ -v /(PATH of the file on which AES have to done)/FILENAME:/AES/(FILENAME) - Map the local dir : container dir for the input data file.

for eg my file is bin.test.bin -v /home/logic-fruit/bin.test:/bin.test,bin

- ✓ \$XILINX_DOCKER_DEVICES - Environment variable set by the host setup script

In the \$XILINX_DOCKER_DEVICES environment variable, list down the device's information that is present on the host machine out to the docker container so that our docker container knows on which device we need to run.

```
echo $XILINX_DOCKER_DEVICES
```

```
--device=/dev/xclmgmt256:/dev/xclmgmt256  
--device=/dev/dri/renderD128:/dev/dri/renderD128  
--device=/dev/dri/renderD129:/dev/dri/renderD129
```

ii. Host Application Details

This is a list of packages that are required to install.

- ✓ XRT
- ✓ Vim
- ✓ g++
- ✓ xcl2.hpp (very important header fill for the Opencl)
- ✓ ocl-icd-libopencl1
- ✓ lsb-release

- ✓ Dkms
- ✓ udev
- ✓ udev:i386
- ✓ python3
- ✓ ocl-icd-ocl-devel
- ✓ uuid-dev
- ✓ libboost-fs1.65.1
- ✓ libboost-program-options1.65.1
- ✓ libboost-system1.65.1
- ✓ libc6
- ✓ libgcc1
- ✓ libncurses5
- ✓ libprotobuf10
- ✓ libssl1.1
- ✓ libstdc++6
- ✓ libtinfo5
- ✓ libudev1
- ✓ libuuid1
- ✓ libyaml-0-2
- ✓ ocl-icd-libopencl1
- ✓ ocl-icd-libopencl1
- ✓ nvidia-libopencl1-340

Environment setup

```
source /opt/xilinx/xrt/setup.sh
```

This will set up the environment variables required for building the host code

```
hitesh@hitesh-Lenovo-V330-14IKB:~$ source /opt/xilinx/xrt/setup.sh
XILINX_XRT      : /opt/xilinx/xrt
PATH            : /opt/xilinx/xrt/bin:/home/hitesh/.local/bin:/usr/local/sbin:
                /usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/b
in
LD_LIBRARY_PATH : /opt/xilinx/xrt/lib:
PYTHONPATH     : /opt/xilinx/xrt/python:
hitesh@hitesh-Lenovo-V330-14IKB:~$
```

Source Code Structure

We have a single folder that contains our source code (main.cpp), Makefile for the compilation, and giving out the executable and the required xcl2.hpp header file. We don't have a modular code in this application.

Logic Implementation

The whole host code is written in C++.

1. The host and kernel code is compiled separately to create separate executable files: the host program executable (APP) and the FPGA binary (.xclbin). When the host application runs, it must load the .xclbin file
2. We provide the Kernel binary (xclbin) file through the command line argument so it is captured into **char* argv[1]** and used by the host applications.
3. Then we initialize 2 vectors to store the data and to get the data back from the FPGA after encryption.
4. Then our host application needs to identify a platform composed of one or more Xilinx devices.
5. After the Xilinx platform is found the application needs to identify the corresponding Xilinx devices.
6. Then we create a context that contains a Xilinx Device that can communicate with the host machine.
7. Then we create a command queue for each device. This command queue is a Single out-of-order command queue which means multiple kernel executions can be requested through the same command queue. XRT dispatches kernels as soon as possible, in any order, allowing concurrent kernel execution on the FPGA.
8. Host application reads the data from the kernel binary file (xclbin) provided in the **char* argv[1]**. And save the data into a character pointer.
9. We take the device handle which is obtained by opening a device. We can pass this device handle to refer to the opened devices in all future interaction with XRT. Function used – ([“xclDeviceHandle xclOpen\(unsigned deviceIndex, const char *logFileName, xclVerbosityLevel level\) “](#)).
10DRM Sessions starts.
10. DRM Sessions starts.
11. After setting up the runtime environment, such as identifying devices, creating the context, command queue, and program, the host application should identify the kernels that will execute on the device, and set up the kernel arguments.
12. Then we should access the kernels contained within the .xclbin file (the "program"), identify a kernel in the program loaded into the FPGA that can be run by the host application.
13. Then three kernels are made: kernel_input, kernel_adder, kernel_output respectively.
14. We set up the kernel arguments as memory buffer arguments that are used for large data transfer. The value is a pointer to a memory object created with the context associated with the program and kernel objects and can be used as inputs to, or outputs from the kernel.

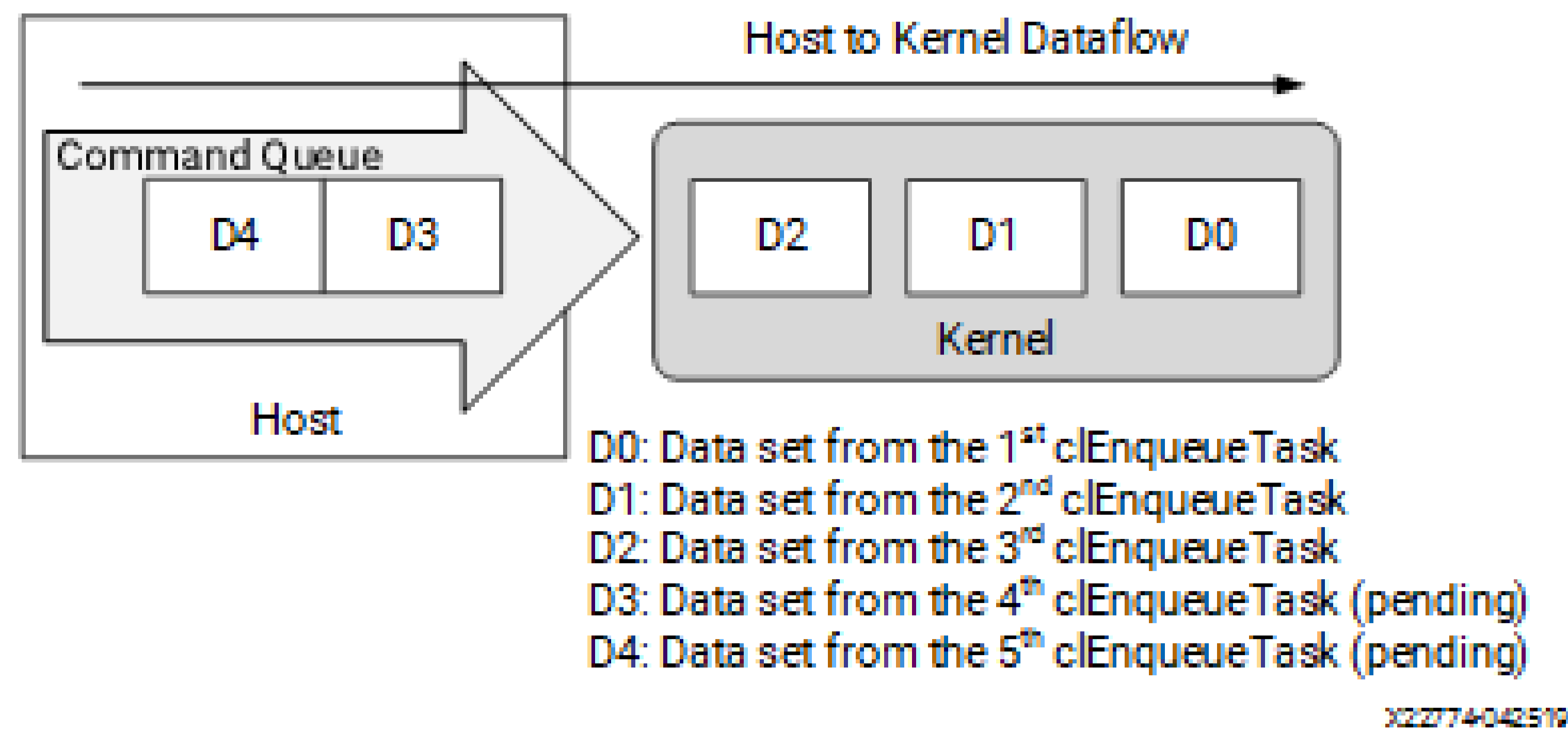
15. Then we make the two buffers `buffer_input` and `buffer_output` as the interactions between the host program and hardware kernels rely on these buffers, transferring data to and from the memory in the device.
16. We need to set up kernel arguments as early as possible as the XRT will error out if we try to migrate the buffer before XRT knows where to put it on the device. Therefore, set the kernel arguments before performing any enqueue operation on any buffer.
17. Then we write data from the host memory to the `buffer_input` using the `enqueueWriteBuffer`, this helps to enable the software pipelining from the host machine to the device buffer. Function used - (`“cl_int clEnqueueWriteBuffer (cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write, size_t offset, size_t cb, const void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)”`).
18. RTL Kernel is executed only one time and works on the entire range of the data, the parallelism is achieved on the FPGA inside the kernel hardware. If properly coded, the kernel is capable of achieving parallelism by various techniques such as instruction-level parallelism (loop pipeline) and function-level parallelism (dataflow). This happens symmetrically.
19. XRT schedules the workload, or the data passed through OpenCL buffers from the kernel arguments, and schedules the kernel tasks to run on the accelerator on the Xilinx FPGA.
20. FPGA then performs the encryption on the data and copies it to the `buffer_output`.
21. Then we read the data from the `buffer_output` and copy it to the localhost machine using the `enqueueReadBuffer`, this helps to enable the software pipelining from the host machine to the device buffer. Function used - (`“cl_int clEnqueueReadBuffer (cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read, size_t offset, size_t cb, void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)”`).
22. We use the `Clfinish()` function which is explicitly used to block the host execution until the kernel execution is finished. This is necessary otherwise the host can attempt to read back from the FPGA buffer too early and may read garbage data.
23. DRM session ends.
24. The final data (Encrypted data) is copied to a file where the user can view it.

Key Steps

Setting Up the Runtime Environment.

1. Getting Platforms.
2. Getting Devices.
3. DRM starts.
4. Setting Up Kernels.
5. Buffer Creation and Data Transfer.
6. Setting Kernel Arguments.
7. Kernel Execution.
8. Event Synchronization.
9. DRM stops.

Flow chart



Data Flow

1. Vectors are initialized.
2. Data from the input file are read and written in those vectors.
3. Two buffers named `input_buffer` and `output_buffer` for input and output are made respectively.
4. The data is copied from the host memory to the buffer using the function
5. Write Function = (`"cl_int cl::CommandQueue::enqueueWriteBuffer(const Buffer& buffer, cl_bool blocking_write, ::size_t offset, ::size_t size, const void * ptr, const VECTOR_CLASS<Event> * events = NULL, Event * event = NULL)"`).
6. Using this `buffer_input` FPGA(alveo u200) read the data from the buffer and performs AES encryption on it. Then FPGA writes the encrypted data to the `buffer_output`.
7. FPGA reads the data from the `buffer_output` and copies the data to host memory.
8. Read Function = (`"cl_int cl::CommandQueue::enqueueReadBuffer(const Buffer& buffer, cl_bool blocking_read, ::size_t offset, ::size_t size, const void * ptr, const VECTOR_CLASS<Event> * events = NULL, Event * event = NULL)"`).
9. Host memory data is then copied to a file for the user to see the encrypted output.

Integration with DRM

1. The DRM in our source code (`main.cpp`) checks for files (`"conf.json"`) and (`"cred.json"`).
2. If both these files are successfully acquired by the DRM, then DRM will activate function (`"void activate(const bool& resume_session_request = false);"`) to start the session, this function activate/unlocks the hardware by unlocking the protected IPs in the FPGA and opening a DRM session. If a session is still pending the behavior depends on the `"resume_session_request"` argument. If true the session is reused. Otherwise, the session is closed and a new session is created. This function will start a thread that keeps the hardware unlocked by automatically updating the license when necessary. When this function returns and the license is valid, the protected IPs are guaranteed to be unlocked.
3. After successfully running the AES256 application, DRM will call the deactivate function (`"void deactivate(const bool& pause_session_request = false);"`) to close the session . This function deactivates/locks the hardware back and closes the session. In this case, the session is kept open for later use. This function will join the thread keeping the hardware unlocked. When the function returns, the hardware is guaranteed to be locked.

Compilation/build Steps

We compile our host code (main.cpp) with a Makefile by running the make command. Makefile provides the important flag to compile the host application and build the executable file (APP).

Docker Container Details

Dependency List

There are some packages we need to be installed.

- ✓ docker
- ✓ libc6
- ✓ libglib2.0-0
- ✓ Libx11-6

Environment setup

1. git clone https://github.com/Xilinx/Xilinx_Base_Runtime.git
2. sudo ~/Xilinx_Base_Runtime/utilities/./docker_install.sh
3. source ~/Xilinx_Base_Runtime/utilities/xilinx_docker_setup.sh

Development Steps

1. We have made a Dockerfile which is simply a text-based script of instructions that is used to create a container image.
2. We have started FROM the image Xilinx/xilinx_runtime_base:alveo-2020.2-Ubuntu-18.04 image. But, since we didn't have that on our machine, that image needed to be downloaded.
3. Then we copy files that we require to build the application like copy host code (main.cpp), Make file, and xcl2.hpp to docker containers.
4. We install the important packages required to run our application like g++, curl, etc to fulfill all the dependencies for making the (APP).
5. Then we build the application as we copy the host code (main.cpp) and Makefile to the container and compile it by running the make all command which gives us the executable file (APP).
6. Then we build the container images in which we will install the important packages required to run our application.
7. We copy the xclbin file (FPGA binary file) which is required to run, the APP is copied from the host machine to the docker container.
8. Then in the Dockerfile we set up some Environment Variables to run the app smoothly.
9. Finally, our Docker container will be able to run the following command to start the application. When the host application runs, it must load the xclbin file.

For Example :-

```
< ./app /xclbin/ABC.xclbin >
```

Integration with host application:

1. In order to build the container we require Dockerfile which is simply a text based script of instructions that is used to create a container image.
2. Our DockerFile is a multi-stage build.
3. In Stage 1 of Dockerfile we build the application as we copy the host code (main.cpp) and Makefile to the container and compile it by running the make all command which gives us an executable file (APP).
4. In Stage 2 of Dockerfile, we build the container image as we copy the xclbin, conf.json, and the APP from the previous stage.
5. By running the “docker run” command the docker container will automatically start and execute the command to run the applications.
6. `< ./app /xclbin/abc.xclbin >`

Container building steps

We have made a multi-stage Dockerfile in which artifacts can be reused from one stage to another stage, leaving behind everything we don't want in our final images. This helps us in increasing the efficiency of the Dockerfile and gives us the benefits of easy to read, easy to maintain and it slims down by reducing some layers in the Dockerfile and reducing the complexity.

Publishing/Hosting

To publish a docker image we need to follow the following steps:

1. Go to the [docker hub](#).
2. Setup your Username and password.
3. Create a repository by giving the name and the description and setting its visibility mode.
4. We need to build the docker image from the docker file and a “context”. Context is a set of the files located in the specified PATH so that if any changes are made it is updated to the new image.

For eg :- `docker build -t hubxilinx/logicfruit_aes256_u200:latest`.

Create Repository

hitesharora97 | Name

Description

Visibility

Using 0 of 1 private repositories. [Get more](#)

Public Appears in Docker Hub search results

Private Only visible to you

Build Settings (optional)

Autobuild triggers a new build with every git push to your source code repository. [Learn More](#)

Connected Disconnected

Pro tip

You can push a new image to this repository using the CLI

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to change tagname with your desired Image repository tag.

- The final step is to share the image on to the docker hub as it is ready for deployment.
For eg :- `docker push hubxilinx/logicfruit_aes256_u200:latest`

Application Usage

The application is containerized and can be easily run in a few minutes on the Alveo card U200.

Runtime Environment

Component	Requirement
Motherboard	PCI Express® 3.0-compatible with one dual-width x16 slot
System Power Supply	225W via PCI Express Slot connection and 8-pin PCI Express Auxiliary Power cable.
Operating System	Linux, 64-bit: <ul style="list-style-type: none"> • Ubuntu 16.04, 18.04 • CentOS 7.4, 7.5, 7.6 • RHEL 7.4, 7.5, 7.6
System Memory	For deployment installations, a minimum of 16 GB plus application memory requirements is required. For development installations, a minimum of 64 GB of device memory is required, but 80 GB is recommended.
Internet Connection	Required for downloading drivers and utilities.
Hard disk space	Satisfy the minimum system requirements for your operating system.
Platform	Alveo U200

Prerequisite

This application supports the Xilinx FPGA Alveo U200 card at this moment. To run this application on users machines, please make sure:

- ☑ For Alveo U200, Xilinx FPGA Alveo U200 (shell `xilinx_u200_xdma_201830_2`) card is installed correctly. (default device id is 0)
- ☑ Docker (with sudo access): When deployed in Nimbix, PushToCompute flow will deploy the application in an instance with `ubuntu18.04`, `U200`, and `XRT 2020.2`.

Useful Xilinx commands:

Command To Check
<code>cat etc/os-release</code>
<code>uname -r</code>

```
vitis -version

sudo lspci -vd 10ee:

sudo /opt/xilinx/xrt/bin/unwrapped/xbmgmt flash --scan --verbose

sudo /opt/xilinx/xrt/bin/unwrapped/xbmgmt flash --scan

docker version
```

iii. Check whether the FPGA is connected and in a working state:

source /opt/xilinx/xrt/setup.sh

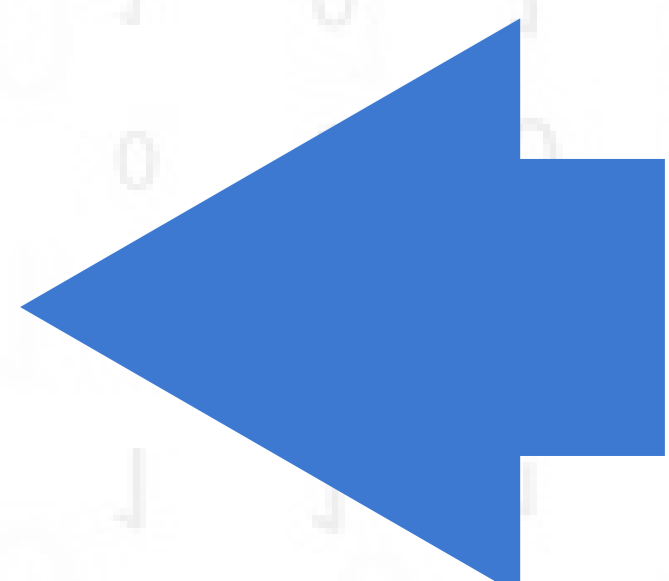
Commands	Usage
xbutil scan	List the card available on the devices , the XRT information & the system configurations
xbutil validate	This command will check the proper functioning of the FPGA on your host machine and even will flash the FPGA card on the host machine
xbutil query	This command will show all the important information about the FPGA card such as card temperature, card memory, power supplied to the card.
xbutil reset	To reset the PL on FPGA.

Chapter 7: Troubleshooting

Successful Build

The successful build of an application is similar to the below message screen.

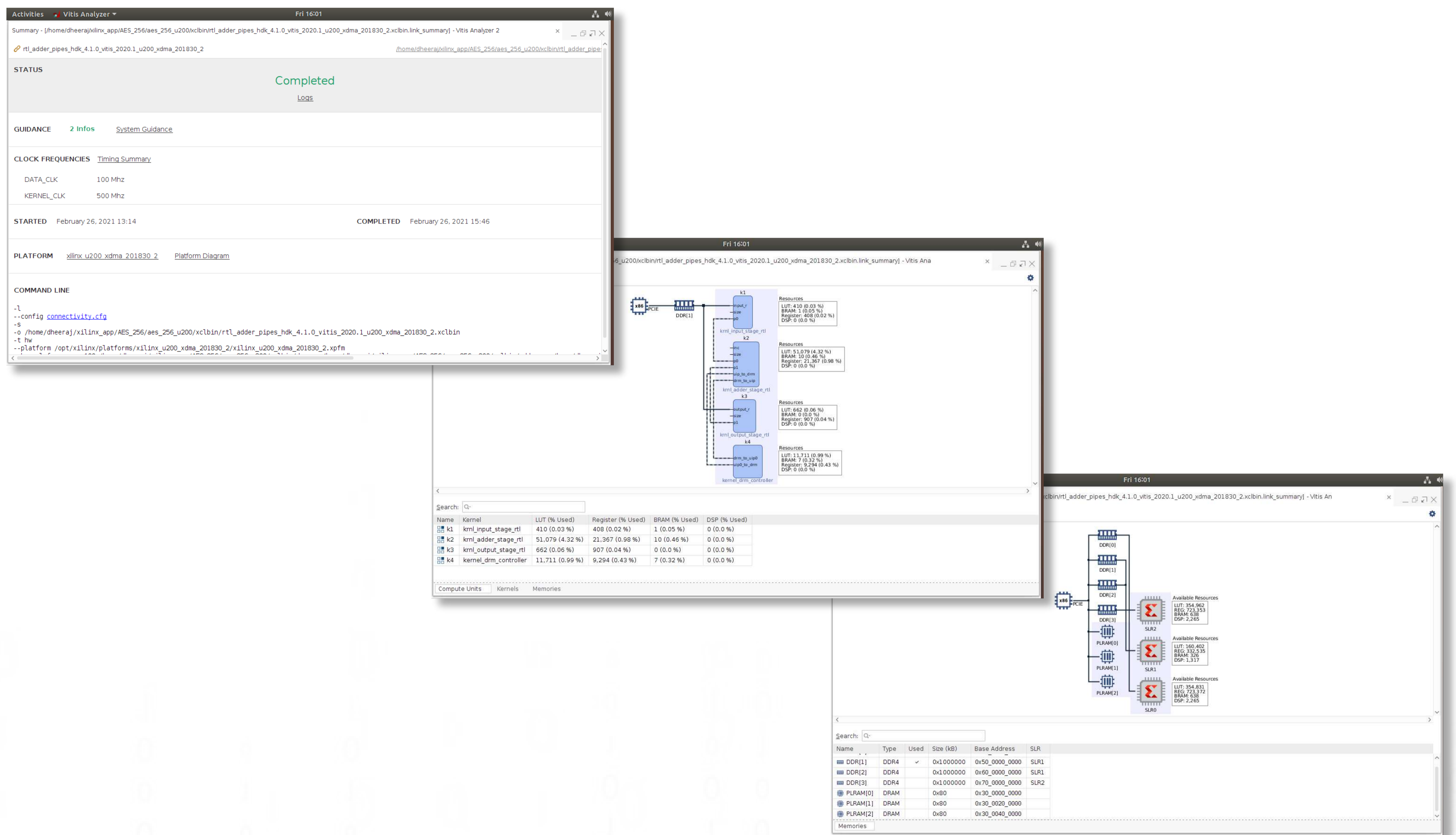
```
dheeraj@dheeraj-Latitude-E5470: ~/xilinx_app/AES_256/aes_256_u200
File Edit View Search Terminal Help
020.1_u200_xdma_201830_2.xclbin
Leaving xclbinutil.
INFO: [v++ 60-1441] [15:46:31] Run run_link: Step xclbinutil: Completed
Time (s): cpu = 00:00:00.13 ; elapsed = 00:00:00.21 . Memory (MB): peak = 1346.414 ; gain = 0.000 ; free physical = 12830 ; free virtual = 14796
INFO: [v++ 60-1443] [15:46:31] Run run_link: Step xclbinutilinfo: Started
INFO: [v++ 60-1453] Command Line: xclbinutil --quiet --force --info /home/dheeraj/xilinx_app/AES_256/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin.info --input /home/dheeraj/xilinx_app/AES_256/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
INFO: [v++ 60-1454] Run Directory: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/link/run_link
INFO: [v++ 60-1441] [15:46:32] Run run_link: Step xclbinutilinfo: Completed
Time (s): cpu = 00:00:00.95 ; elapsed = 00:00:00.01 . Memory (MB): peak = 1346.414 ; gain = 0.000 ; free physical = 12830 ; free virtual = 14796
INFO: [v++ 60-1443] [15:46:32] Run run_link: Step generate_sc_driver: Started
INFO: [v++ 60-1453] Command Line:
INFO: [v++ 60-1454] Run Directory: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/link/run_link
INFO: [v++ 60-1441] [15:46:32] Run run_link: Step generate_sc_driver: Completed
Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.01 . Memory (MB): peak = 1346.414 ; gain = 0.000 ; free physical = 12830 ; free virtual = 14796
INFO: [v++ 60-244] Generating system estimate report...
INFO: [v++ 60-1092] Generated system estimate report: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/reports/link/system_estimate_rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.txt
INFO: [v++ 60-586] Created /home/dheeraj/xilinx_app/AES_256/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.ltx
INFO: [v++ 60-586] Created /home/dheeraj/xilinx_app/AES_256/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
INFO: [v++ 60-1307] Run completed. Additional information can be found in:
Guidance: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/reports/link/v++_link_rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2_guidance.html
Timing Report: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/reports/link/imp/xilinx_u200_xdma_201830_2_bb_locked_timing_summary_routed.rpt
Utilizations Report: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/reports/link/imp/kernel_util_routed.rpt
Vivado Log: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/logs/link/vivado.log
Steps Log File: /home/dheeraj/xilinx_app/AES_256/aes_256_u200/_x/logs/link/link.steps.log
INFO: [v++ 60-2343] Use the vitis_analyzer tool to visualize and navigate the relevant reports. Run the following command.
vitis_analyzer /home/dheeraj/xilinx_app/AES_256/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin.link_summary
INFO: [v++ 60-791] Total elapsed time: 2h 32m 31s
INFO: [v++ 60-1653] Closing dispatch client.
dheeraj@dheeraj-Latitude-E5470:~/xilinx_app/AES_256/aes_256_u200$
```



Use Vitis Analyzer tool to visualize and navigate reports

Use command: `vitis_analyzer <path_to>/xclbin/<file_name>.xclbin.link_summary`

Some of the analyzer reports are attached here for reference.



[Error] Unable to find DRM controller registers

Could not access DRM controller registers.

```
INFO: Reading /home/user/xilinx_app/GettingStarted_Examples/Hardware/Xilinx_Vitis/rtl_adder_pipes_Alveo/
xclbin/rtl_adder_pipes_hdk_4.2.1_vitis_2020.1.xclbin done!
[error] 2685 , Unable to find DRM Controller registers.
Could not access DRM Controller registers.
Please verify:
  -The read/write callbacks implementation in the SW application: verify it uses the correct offset
  t address of DRM Controller IP in the design address space.
  -The DRM Controller IP instantiation in the FPGA design: verify the correctness of 16-bit address
  s received by the AXI-Lite port of the DRM Controller.
[critical] 2685 , [errCode=20001] Unable to find DRM Controller registers.
Could not access DRM Controller registers.
Please verify:
  -The read/write callbacks implementation in the SW application: verify it uses the correct offset
  t address of DRM Controller IP in the design address space.
  -The DRM Controller IP instantiation in the FPGA design: verify the correctness of 16-bit address
  s received by the AXI-Lite port of the DRM Controller.
terminate called after throwing an instance of 'Accelize::DRM::Exception'
  what(): [errCode=20001] Unable to find DRM Controller registers.
Could not access DRM Controller registers.
Please verify:
  -The read/write callbacks implementation in the SW application: verify it uses the correct offset
  t address of DRM Controller IP in the design address space.
  -The DRM Controller IP instantiation in the FPGA design: verify the correctness of 16-bit address
  s received by the AXI-Lite port of the DRM Controller.
Aborted (core dumped)
```

The error is due to the mismatch between the initial offset memory page of the DRM controller registers and the one defined in the host application <main.cpp>. The Vitis platform generates this address unique to the initial build for the application or if the HDK package is updated/changed. It remains the same if set once, But it is advised to check if the error persists.

Open the file <file_name_hdk_version_vitis_version_xdma_platformu>.xclbin.info i.e, rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin.info located in the output directory xclbin.

The file contains information about the xclbin generated and the Hardware Platform (Shell). Traverse to the kernel instance section, locate the kernel instance for the DRM controller IP, use the base address from it, and replace it in the host app <main.cpp> for DRM_BASE_ADDRESS.

[Error] Unable to find DRM controller registers

```
Instance:      k4
Base Address: 0x1c30000

Argument:     drm_to_uip0
Register Offset: 0x0
Port:         drm_to_uip0
Memory:       dc_3 (MEM_STREAMING_CONNECTION)

Argument:     uip0_to_drm
Register Offset: 0x0
Port:         uip0_to_drm
Memory:       dc_2 (MEM_STREAMING_CONNECTION)
```

```
12 #define OCL_CHECK(error,call)
13 call,
14 if (error != CL_SUCCESS) {
15     printf("%s:%d Error calling " #call ", error code is: %d\n", \
16           __FILE__, __LINE__, error);
17     exit(EXIT_FAILURE);
18 }
19
20 #define DATA_SIZE 4092
21 #define INCR_VALUE 10
22 #define REG_DATA 8
23 #define DRM_BASE_ADDRESS 0x1c30000
24
```

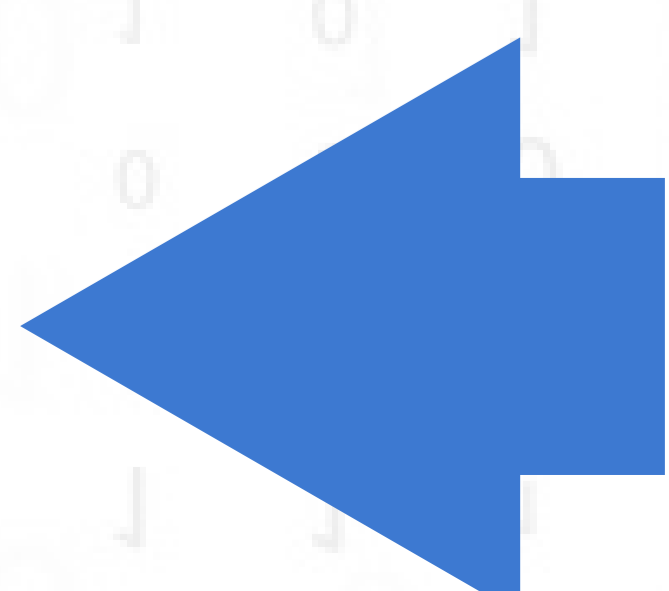
Reference links

https://tech.accelize.com/documentation/stable/drm_troubleshooting.html

[Error] Path is not a valid file: cred.json

The error is due to the invalid license file path defined in the conf.json file. Both files are located in the directory /app. Update the correct file path of the cred.json in the file conf.json.

```
user@user-To-be-filled-by-0-E-M:~/xilinx_app_store/aes_256_u200/app$ ./app ~/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
INFO: Reading /home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
Loading: '/home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin'
INFO: Reading /home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin done!
[error] 15273 , Path is not a valid file: cred.json
[error] 15273 , Error with credential file 'cred.json': [errCode=1] Path is not a valid file: cred.json
[critical] 15273 , [errCode=1] Error with credential file 'cred.json': [errCode=1] Path is not a valid file: cred.json
terminate called after throwing an instance of 'Accelize::DRM::Exception'
what(): [errCode=1] Error with credential file 'cred.json': [errCode=1] Path is not a valid file: cred.json
Aborted (core dumped)
```




```

{
  "design": {
    "boardType": "Alveo_U200"
  },
  "licensing": {
    "url": "https://master.metering.accelize.com"
  },
  "drm": {
    "frequency_mhz": 180,
    "mode": "metering",
    "license_dir": "~/xilinx_app_store/aes_256_u200/app"
  },
  "settings": {
    "log_verbosity": 2
  }
}

```

[Error] Metering web service error 400: User account has no entitlement

The error is due to the unavailability of license entitlement to the user. Also, it might be that the user is entitled to use the app but hasn't subscribed to the plan or the subscription has expired. Please check and update your subscription plans for the app in your Accelize account.

Reference Links:

https://tech.accelize.com/documentation/stable/drm_trouble-shooting.html#if-you-get-this-error-message-drm-ws-request-failed

```

user@user-To-be-filled-by-0-E-M:~/xilinx_app_store/new_folder/rtl_adder_pipes_Alveo/app$ ./app ~/xilinx_
app_store/new_folder/rtl_adder_pipes_Alveo/xclbin/rtl_adder_pipes_hdk_4.2.1_vitis_2020.1.xclbin
INFO: Reading /home/user/xilinx_app_store/new_folder/rtl_adder_pipes_Alveo/xclbin/rtl_adder_pipes_hdk_4.
2.1_vitis_2020.1.xclbin
Loading: '/home/user/xilinx_app_store/new_folder/rtl_adder_pipes_Alveo/xclbin/rtl_adder_pipes_hdk_4.2.1_
vitis_2020.1.xclbin'
INFO: Reading /home/user/xilinx_app_store/new_folder/rtl_adder_pipes_Alveo/xclbin/rtl_adder_pipes_hdk_4.
2.1_vitis_2020.1.xclbin done!
[DRMLIB] Start Session ..
[error] 15625 , Metering Web Service error 400: {"error":true,"detail":"DRM WS request failed. \No E
ntitlement\" with [GettingStarted] 01_rtl_kernel for dheeraj.punia@logic-fruit.com : User account has no
entitlement. Purchase additional licenses via your portal."}
[critical] 15625 , [errCode=10002] Metering Web Service error 400: {"error":true,"detail":"DRM WS reques
t failed. \No Entitlement\" with [GettingStarted] 01_rtl_kernel for dheeraj.punia@logic-fruit.com : Use
r account has no entitlement. Purchase additional licenses via your portal."}
DRM error: [errCode=10002] Metering Web Service error 400: {"error":true,"detail":"DRM WS request failed
. \No Entitlement\" with [GettingStarted] 01_rtl_kernel for dheeraj.punia@logic-fruit.com : User accoun
t has no entitlement. Purchase additional licenses via your portal."}
XRT build version: 2.8.743

```

[Error] Metering web service error 400

The collection of Activators from the license request does not match the expected configuration.

The error is due to the mismatch between the DRM Activators integrated with the app and the Accelize web portal. Please check and update the DRM HDK package version integrated with the application and the webserver.

```
INFO: Reading /home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin done!
[DRMLIB] Start Session ..
[error] 21047 , Metering Web Service error 400: {"error":true,"detail":"Invalid Product Configuration"} with [GettingStarted] 01_rtl_kernel for N/A: the configuration for [GettingStarted] 01_rtl_kernel is invalid. The collection of Activators from the license request does not match the expected configuration. Please contact the application vendor."}
[critical] 21047 , [errCode=10002] Metering Web Service error 400: {"error":true,"detail":"DRM WS request failed. \Invalid Product Configuration\ with [GettingStarted] 01_rtl_kernel for N/A: the configuration for [GettingStarted] 01_rtl_kernel is invalid. The collection of Activators from the license request does not match the expected configuration. Please contact the application vendor."}
DRM error: [errCode=10002] Metering Web Service error 400: {"error":true,"detail":"DRM WS request failed. \Invalid Product Configuration\ with [GettingStarted] 01_rtl_kernel for N/A: the configuration for [GettingStarted] 01_rtl_kernel is invalid. The collection of Activators from the license request does not match the expected configuration. Please contact the application vendor."}
XRT build version: 2.8.743
Build hash: 77d5484b5c4daa691a7f78235053fb036829b1e9
```

[XRT] Error: CU was deadlocked? Hardware is not stable

The error is due to the CU unit or Programmable Logic (PL) on the Alveo board/hardware being unstable or hang. Use the XRT command to reset the PL surface on the board. Use the below command

xbutil reset

If the error still persists, try a cold reboot of the host PC. Also, check the XRT version installed. Update it if it is corrupt or outdated.

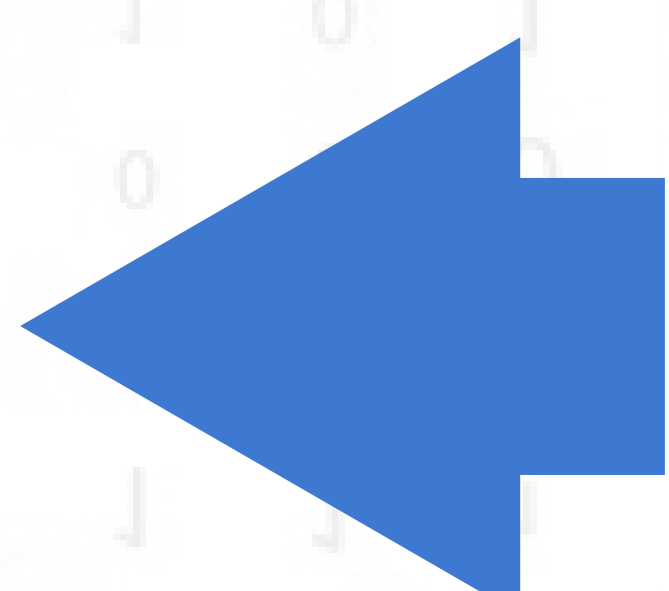
Reference Links:

<https://forums.xilinx.com/t5/Alveo-Accelerator-Cards/-First-Alveo-U280-kernel-run-XRT-ERROR-No-devices-found/td-p/1047055>

There are various other xbutil commands to help in runtime and debug.

https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/xilinx-board-swiss-army-knife-utility-ufa1504034339078.html

```
INFO: Reading /home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin done!
XRT build version: 2.8.743
Build hash: 77d5484b5c4daa691a7f78235053fb036829b1e9
Build date: 2020-11-16 00:19:11
Git branch: 2020.2
PID: 21165
UID: 1000
[Thu Mar 11 06:47:38 2021 GMT]
HOST: user-To-be-filled-by-O-E-M
EXE: /home/user/xilinx_app_store/aes_256_u200/app/app
[XRT] ERROR: CU was deadlocked? Hardware is not stable
[XRT] ERROR: Please reset device with 'xbutil reset'
[XRT] ERROR: See dmesg log for details. err=-35
[XRT] ERROR: Failed to load xclbin.
main.cpp:163 Error calling cl::Program program(context, devices, bins, NULL, &err), error code is: -44
user@user-To-be-filled-by-O-E-M:~/xilinx_app_store/aes_256_u200/app$
```



[Error] Bus Interface property FREQ_HZ does not match between <port_1> and <port_2>

The error is due to a clock frequency mismatch between the two ports defined. Check the operating frequency in the makefile and the kernel operating frequency defined. Also, do check the tickle scripts defined in the src folder for the clock and reset signals declaration integration between the DRM controller IP and the kernels using the port signals accordingly.

Reference Links:

<https://forums.xilinx.com/t5/Processor-System-Design-and-AX-I/BD-41-237-Bus-Interface-property-FREQ-HZ-does-not-match/td-p/775283>

<https://www.xilinx.com/support/answers/56610.html>

<https://forums.aws.amazon.com/thread.jspa?threadID=271665>

```
====>The following messages were generated while creating FPGA bitstream. Log file: /home/user/xilinx_app_store/aes_256_u200/_x/link/vivado/vpl/runme.log :
ERROR: [VPL 41-237] Bus Interface property FREQ_HZ does not match between /k4/uip0_to_drm(500000000) and /k2/uip_to_drm(300000000)
ERROR: [VPL 41-237] Bus Interface property CLK_DOMAIN does not match between /k4/uip0_to_drm(pfm_dynamic_clkwiz_kernel2_clk_out1) and /k2/uip_to_drm(pfm_dynamic_clkwiz_kernel_clk_out1)
ERROR: [VPL 41-237] Bus Interface property FREQ_HZ does not match between /k2/drm_to_uip(300000000) and /k4/drm_to_uip0(500000000)
ERROR: [VPL 41-237] Bus Interface property CLK_DOMAIN does not match between /k2/drm_to_uip(pfm_dynamic_clkwiz_kernel_clk_out1) and /k4/drm_to_uip0(pfm_dynamic_clkwiz_kernel2_clk_out1)
ERROR: [VPL 41-1031] HdL Generation failed for the IP Integrator design /home/user/xilinx_app_store/aes_256_u200/_x/link/vivado/vpl/prj/prj.srscs/my_rn/bd/bd/pfm_dynamic.bd
```

[XRT] Warning: unaligned host pointer '0x7fffxxxxxx' detected, this lead to extra memcpy

To remove the alignment warning, you need to use the "aligned_allocator" that Xilinx provides in their lib XCL2. These are some of the header files and their functions we require to use in order to remove these warnings. These warnings will not have any effect on the functionality of the application yet it is better to remove if any.

Reference Links:

<https://developer.xilinx.com/en/articles/example-2-aligned-memory-allocation.html>

<https://forums.xilinx.com/t5/Vitis-Acceleration-SDAccel-SD-SoC/memory-alignment-when-allocating-ememory-in-SDAccel/td-p/887593>

```

user@user-To-be-filled-by-0-E-M:~/xilinx_app_store/aes_256_u200/app$ ./app ~/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
INFO: Reading /home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
Loading: '/home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin'
INFO: Reading /home/user/xilinx_app_store/aes_256_u200/xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin done!
[DRMLIB] Start Session ..
[ info ] 16158 , DRM session C0FA1E94481CED8C created.
XRT build version: 2.8.743
Build hash: 77d5484b5c4daa691a7f78235053fb036829b1e9
Build date: 2020-11-16 00:19:11
Git branch: 2020.2
PID: 16158
UID: 1000
[Tue Mar 23 11:10:13 2021 GMT]
HOST: user-To-be-filled-by-0-E-M
EXE: /home/user/xilinx_app_store/aes_256_u200/app/app
[XRT] WARNING: unaligned host pointer '0x7fff40e45550' detected, this leads to extra memcpy
[XRT] WARNING: unaligned host pointer '0x7fff40e49550' detected, this leads to extra memcpy
Bus error (core dumped)

```

[XRT] Error: Cannot add a component to the argument

The error is due to the unsatisfied properties of the vitis kernel requirement. Refer to the web page and follow as described. There might be a mismatch between register offset address, ports, or signal usage.

The kernel ports, signals, and arguments are defined in the RTL and the XML files for each kernel. All the definitions and usage should match for a successful build.

Reference Links:

https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/devrtlkernel.html

https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/myl1532064542647.html#:~:text=An%20XML%20kernel%20description%20file,runtime%20and%20Vitis%20tool%20flows.

https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/rtl_kernel_wizard.html

```

[DRMLIB] Start Session ..
[ info ] 30551 , DRM session B36153FBB2E701DE created.
XRT build version: 2.8.743
Build hash: 77d5484b5c4daa691a7f78235053fb036829b1e9
Build date: 2020-11-16 00:19:11
Git branch: 2020.2
PID: 30551
UID: 1000
[Thu Apr 15 05:03:40 2021 GMT]
HOST: user-To-be-filled-by-0-E-M
EXE: /home/user/xilinx_app_store/old_builds_v1/new_hdk_build/bus_interface_freq_HZ/aes_256_u200/app/app
[XRT] ERROR: Cannot add component to argument
main.cpp:201 Error calling cl::Kernel krnl_adder_stage(program,"krnl_adder_stage_rtl", &err), error code is: -42

```

Check md5sum value of the <file_name>.xclibin

Because nearly every modification to a file will cause its MD5 hash to change, md5sum is used to verify the integrity of files. Md5sum is most typically used to ensure that a file has not been altered due to a failed file transfer, a disk malfunction, or non-malicious tinkering. Check md5sum using the below command:

`md5sum filename`

```
(base) user@user-To-be-filled-by-0-E-M:~/xilinx_app_store/old_builds_v1/new_hdk_build/bus_interface_freq_HZ/aes_256_u200_32bit/app$ md5sum ../xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
fca66c1f1074f1b9f40815aa656e4c51  ../xclbin/rtl_adder_pipes_hdk_4.1.0_vitis_2020.1_u200_xdma_201830_2.xclbin
```

The first step is you can see which devices are present on your host

`sudo lspci -vd 10ee:`

```
test@test-To-be-filled-by-0-E-M:~$ sudo lspci -vd 10ee:
[sudo] password for test:
01:00.0 Processing accelerators: Xilinx Corporation Device 5000
Subsystem: Xilinx Corporation Device 000e
Flags: bus master, fast devsel, latency 0
Memory at f2000000 (64-bit, prefetchable) [size=32M]
Memory at f4000000 (64-bit, prefetchable) [size=128K]
Capabilities: [40] Power Management version 3
Capabilities: [60] MSI-X: Enable+ Count=33 Masked-
Capabilities: [70] Express Endpoint, MSI 00
Capabilities: [100] Advanced Error Reporting
Capabilities: [1c0] #19
Capabilities: [400] Access Control Services
Capabilities: [410] #15
Kernel driver in use: xclmgmt
Kernel modules: xclmgmt

01:00.1 Processing accelerators: Xilinx Corporation Device 5001
Subsystem: Xilinx Corporation Device 000e
Flags: bus master, fast devsel, latency 0, IRQ 16
Memory at f0000000 (64-bit, prefetchable) [size=32M]
Memory at f4020000 (64-bit, prefetchable) [size=64K]
Memory at e0000000 (64-bit, prefetchable) [size=256M]
Capabilities: [40] Power Management version 3
Capabilities: [60] MSI-X: Enable+ Count=33 Masked-
Capabilities: [70] Express Endpoint, MSI 00
Capabilities: [100] Advanced Error Reporting
Capabilities: [400] Access Control Services
Capabilities: [410] #15
Kernel driver in use: xocl
Kernel modules: xocl
```

This should be the required output if not please do reinstall check the Troubleshooting page.

Determine Linux release:

Use the `cat /etc/*release` command to determine the Linux release

```
File Edit View Search Terminal Help
hitesh@hitesh-Lenovo-V330-14IKB:~$ cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.5 LTS"
NAME="Ubuntu"
VERSION="18.04.5 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.5 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
hitesh@hitesh-Lenovo-V330-14IKB:~$
```

We can review the shell capabilities with by command `sudo /opt/Xilinx/xrt/bin/xbmgmt flash --scan` as shown below

```
File Edit View Search Terminal Help
^[[Atest@test-To-be-filled-by-0-E-M:~$ sudo /opt/xilinx/xrt/bin/xbmgmt flash --scan
Card [0000:01:00.0]
  Card type:          u200
  Flash type:         SPI
  Flashable partition running on FPGA:
    xilinx_u200_xdma_201830_2,[ID=0x5d1211e8],[SC=4.2.0]
  Flashable partitions installed in system:
    xilinx_u200_xdma_201830_2,[ID=0x5d1211e8],[SC=4.2.0]

test@test-To-be-filled-by-0-E-M:~$
```

Unload/reload XRT drivers:

Use `modprobe -r` to remove the drivers as shown below

```
sudo modprobe -r xocl
```

```
sudo modprobe -r xclmgmt
```

Use `modprobe` to reload the drivers as shown below

```
sudo modprobe xclmgmt sudo modprobe xocl
```

Order matters for both of these commands. `xocl` depends on `xclmgmt`.

Flash the card with a deployment platform:

```
test@test-To-be-filled-by-0-E-M:/opt/xilinx/xrt/bin/unwrapped$ sudo ./xbmgmt flash --scan
Card [0000:01:00.0]
  Card type:          u200
  Flash type:         SPI
  Flashable partition running on FPGA:
    xilinx_u200_GOLDEN_5,[SC=INACTIVE]
  Flashable partitions installed in system:
    xilinx_u200_xdma_201830_2,[ID=0x5d1211e8],[SC=4.2.0]
```

Once the card is up and running in the system, a deployment platform will need to be flashed onto the card before xutil validate passes and applications can be run. To flash the card with a deployment platform follow the below steps:

- ④ Run `sudo xbmgmt flash --scan`
- ④ If Flashable partitions installed in the system: (None) is the output please install the latest packages from the Alveo landing page for your installed card(s)
- ④ Follow the process for Card install to install the platforms on the machine.
- ④ Run `sudo xbmgmt flash --update --shell <xilinx_uxx>` to flash the platform onto the card. This command should be provided during platform installation, shown below:

```
Partition package installed successfully.
```

```
Please flash card manually by running below command:
```

```
sudo /opt/xilinx/xrt/bin/xbmgmt flash --update --shell xilinx_u200_xdma_201830_2
```

```
~]$ sudo xbmgmt flash --update --shell xilinx_u200_xdma_201830_2
```

```
Status: shell needs updating
```

```
Current shell: xilinx_u200_GOLDEN_9
```

```
Shell to be flashed: xilinx_u200_xdma_201830_2
```

```
Are you sure you wish to proceed? [y/n]: y
```

```
Updating shell on card[0000:05:00.0]
```

```
INFO: ***Found 353 ELA Records
```

```
Enabled bitstream guard. Bitstream will not be loaded until flashing is finished.
```

```
Preparing flash chip 0
```

```
Erasing flash.....
```

```
Programming flash.....
```

```
Cleared the bitstream guard. Bitstream now active.
```

```
Successfully flashed Card[0000:05:00.0]
```

```
1 Card(s) flashed successfully.
```

```
Cold reboot machine to load the new image on card(s).
```

- ✓ Cold boot the server
- ✓ Run `sudo xbmgmt flash --scan`
- ✓ Now platform installed in host and card are the same
- ✓ If this is a DFX-2RP platform, go to Programming DFX-2RP shell partitions
- ✓ If there is a different number in the SC= line between the FPGA and the system for the platform on the card, update the SC firmware, example below:

```

:~> sudo xbmgmt flash --update

Status: SC needs updating

Current SC: 5.0.20

SC to be flashed: 5.0.27

Updating SC firmware on card[0000:05:00.0]

Stopping user function...

INFO: found 4 sections

.....

INFO: Loading new firmware on SC

Successfully flashed Card[0000:05:00.0]

1 Card(s) flashed successfully.

```

Reverting the card to factory image:

The Alveo card can be reverted to the factory image, also known as golden. This requires that XRT release 2019.2 or later is installed on the same system as the Alveo accelerator card. The steps to revert the card using this method are listed below.

- 1.1. Open a terminal window.
- 1.2. Run the following command, where `card_bdf` is the BDF of the card to revert to golden.

```
$ sudo xbmgmt flash --factory_reset --card <card_bdf>
```

- 1.3. Enter `y` to continue. The following message is displayed on completion.

Shell is reset successfully

Cold reboot machine to load new shell on card

1.4.Cold boot the system so the card FPGA uses the new image.

1.5.Confirm the card has been reverted to factory image by running the following command.

```
$ sudo xbmgmt flash --scan
```

1.6.An output similar to the following is displayed.

```
Card [0000:65:00.0]

Card type: uxx

Flash type: SPI

Flashable partition running on FPGA:

xilinx_uxx_GOLDEN_x,[SC=x.x]

Flashable partitions installed in system: (None)
```

In this output, under the Flashable partition running on an FPGA, note GOLDEN in the name. This indicates that the card has successfully been reverted to the factory image.

IMPORTANT! If the GOLDEN_2 image is running on the FPGA, carefully review the design advisory for Alveo data center Accelerator card golden corruption, found in AR 71915. Complete the repair instructions associated with the Xilinx Answer prior to proceeding.

For more information you can log on to this url :

<https://xilinx.github.io/Alveo-Cards/master/debugging/README.html>



Thank You!

Does anyone have any questions?

Contact Us



Gurugram (Headquarter)

806, 8th Floor
BPTP Park Centra Sector-30,
NH-8 Gurgaon - 122001
Haryana (India)

info@logic-fruit.com

+91-0124 4643950



Bengaluru (R&D House)

Sy. No 118, 3rd Floor,
Gayathri Lakefront,
Outer Ring Road, Hebbal,
Bangalore - 560 024

sales@logic-fruit.com

+91 80-69019700/01



United States (Sales Office)

Logic Fruit Technologies
INC 691 S Milpitas Blvd Ste
217 (Room 9) Milpitas
CA 95035

info@logic-fruit.com

+1-408 338 9743