ALIBABA CLOUD

Alibaba Cloud

FPGA as a Service

Best Practices

Document Version: 20220711

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Best practices for RTL design on an FPGA-based ECS instance

1.1. Use RTL Compiler on an f1 instance

This topic describes how to use Register Transfer Level (RTL) Compiler on an f1 instance.

Prerequisites

-
-
-
-
-
-
-

Context

Before you perform the operations, take note of the following items:

Procedure

1. Connect to an f1 instance.

   For more information, see Connect to a Linux instance by using a username and password.
2. Run the following command to configure the basic environment:

   ```
   source /opt/dcp1_1/script/f1_env_set.sh
   ```
3. Run the following commands in sequence to compile the project:

   ```
   cd /opt/dcp1_1/hw/samples/dma_afu
   ```

   ```
   afu_synth_setup --source hw/rtl/filelist.txt build_synth
   ```

   ```
   cd build_synth/
   ```

   ```
   run.sh
   ```

   Note The compilation process may take some time.
4. Create an image.

i. Run the following commands in sequence to configure the `PATH` environment variable and grant execute permissions to the *faascmd* file:

```
export PATH=$PATH:/opt/dcp1_1/script/
```

```
chmod +x /opt/dcp1_1/script/faascmd
```

ii. Run the following commands in sequence to initialize the **faascmd** configuration:

```
# Replace hereIsYourSecretId with your AccessKey ID. Replace hereIsYourSecretKey wi
th your AccessKey secret.
faascmd config --id=hereIsYourSecretId --key=hereIsYourSecretKey
```

```
# Replace hereIsYourBucket with the OSS bucket name in the China (Hangzhou) region.
faascmd auth --bucket=hereIsYourBucket
```

iii. Run the following command in the */opt/dcp1_1/hw/samples/dma_afu* directory to upload the GBS file:

```
faascmd upload_object --object=dma_afu.gbs --file=dma_afu.gbs
```

iv. Run the following command to create an image:

```
# Replace hereIsYourImageName with your image name.
faascmd create_image --object=dma_afu.gbs --fpgatype=intel --name=hereIsYourImageNa
me --tags=hereIsYourImageTag --encrypted=false --shell=V1.1
```

5. Download the image.

i. Run the following command to check whether the image is created:

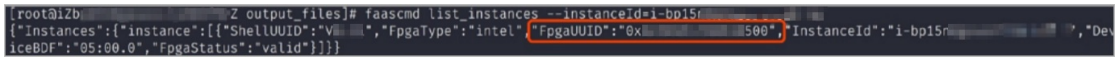
```
faascmd list_images
```

If `"State":"success"` is displayed in the command output, the image is created. Record the value of FpgaImageUUID in the command output for subsequent use.

ii. Run the following command to obtain the ID of your FPGA instance:

```
# Replace hereIsYourInstanceId with the ID of your f1 instance.
faascmd list_instances --instanceId=hereIsYourInstanceId
```

The following figure shows the command output. Record the value of FpgaUUID.



iii. Run the following command to download the FPGA image to the f1 instance:

```
# Replace hereIsYourInstanceID with the instance ID that you recorded. Replace here
IsFpgaUUID with the value of FpgaUUID that you recorded. Replace hereIsImageUUID wi
th the value of FpgaImageUUID that you recorded.
faascmd download_image --instanceId=hereIsYourInstanceID --fpgauuid=hereIsFpgaUUID
--fpgatype=intel --imageuuid=hereIsImageUUID --imagetype=afu --shell=V1.1
```

iv. Run the following command to check whether the image is downloaded:

```
# Replace hereIsYourInstanceID with the instance ID that you recorded. Replace here
IsFpgaUUID with the value of FpgaUUID that you recorded.
faascmd fpga_status --instanceId=hereIsYourInstanceID --fpgauuid=hereIsFpgaUUID
```

If `"TaskStatus":"operating"` is displayed in the command output, and the value of FpgaImageUUID is the same as the value of FpgaImageUUID, the image is downloaded.

6. (Optional)If HugePages is disabled, run the following command to enable HugePages:

```
sudo bash -c "echo 20 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages"
```

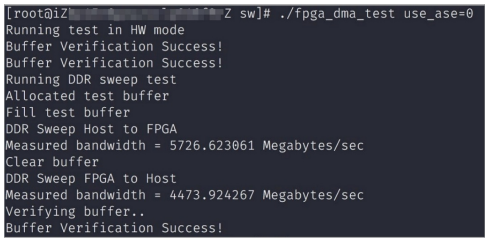
7. Run the following commands in sequence to perform a test:

```
cd /opt/dcp1_1/hw/samples/dma_afu/sw
```

```
make
```

```
sudo LD_LIBRARY_PATH=/opt/dcp1_1/hw/samples/dma_afu/sw:$LD_LIBRARY_PATH ./fpga_dma_test
0
```

If the following command output is displayed, the test is complete.



1.2. Project modes and directories used by RTL

This topic describes the project modes and directories used by the Register Transfer Level (RTL) compiler. This topic also provides a sample framework to help you understand how to use RTL.

Project modes

Vivado Design Suite is an integrated design environment released by a field-programmable gate array (FPGA) vendor. Vivado provides two use modes: Project Mode and NoProject Mode. The RTL compiler on F3 instances uses the NoProject mode. The following section describes the two use modes:

- Project Mode

  In Project Mode, a .bit file is created from a project. In this mode, you must import the RTL code and constraint files to the project before you can generate the .bit file.

- NoProject Mode

In NoProject Mode, a .bit file is created from a .dcp file. In this mode, you must combine all the .dcp files into one .dcp file and configure the connections before you can generate the .bit file.

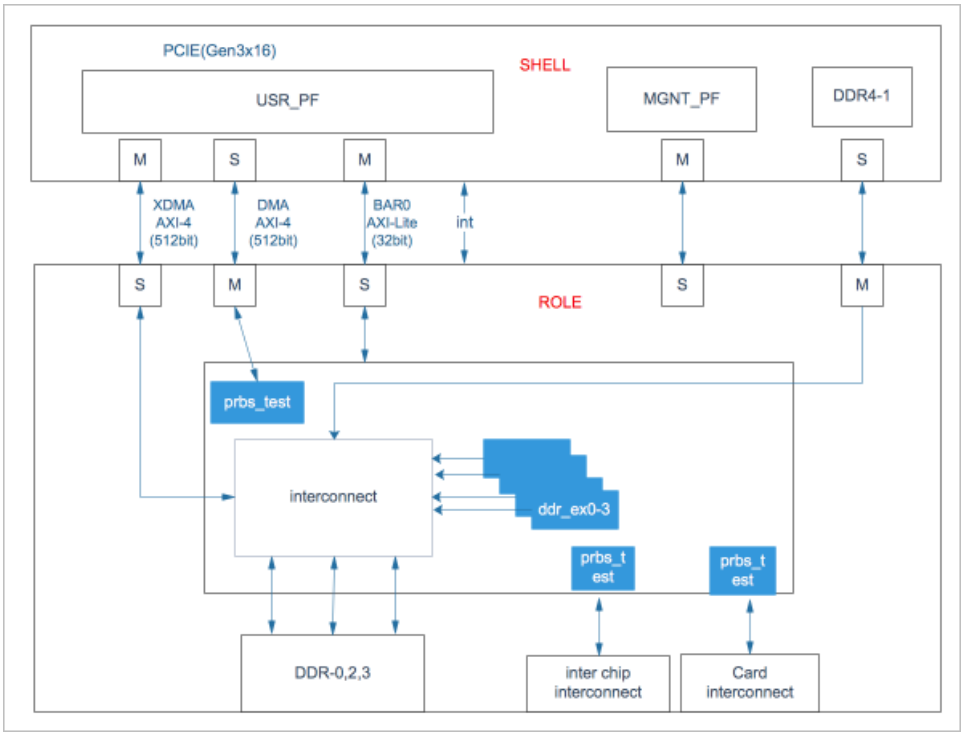
Main directory of a project

The directories of a project include a main directory and a source directory. The project files of the Partial Reconfiguration (PR) section are stored in the source directory, whereas the script files that run the project are stored in the main directory. The following table describes the scripts that are stored in the main directory.

Script	Description
compiling.sh	This script runs the entire project and generates the xclbin load file.
create_design.tcl	This script can be executed in the Tcl Console of Vivado Design Suite. You must use this script to create the usr_top.v top project. Then, you must execute the run_synth.tcl script to call the create_design.tcl script and generate the custom_logic.dcp file. You must add the custom_logic.dcp files to the create_design.tcl script.
mem_design.tcl	This script is executed in the Tcl Console of Vivado Design Suite to re-instantiate the intellectual property (IP) core files of double data rate (DDR) memory. This script must be used together with the config.v file. Three DDR controllers are configured in the dynamic loading area. Configure mem_design.tcl based on the actual hardware you are using to instantiate the actual DDR memory.
run_synth.tcl	This script must be used together with the create_design.tcl script.
run.tcl	This script is called by the compiling.sh script to run the entire project.
generate_dcp.tcl	This script combines multiple .dcp files into one .dcp file to generate the final .dcp and .bit files. You can modify the policy to generate the final .dcp and .bit files.

Sample framework

Alibaba Cloud provides sample code based on the following framework. For more information about the code, see Xilinx/dma_ip_drivers.



- You can add `{ PCI_DEVICE(0x1ded, 0x1004), },` to the following position in the sample code to change the ID of the device. Then, you can use the tool in the device to test your project. For more information about the AR65444 software and its tools, visit the Xilinx official website.

```
#ifdef INTERNAL_TESTING
        { PCI_DEVICE(0x1d0f, 0x1042), 0},
#endif
        /* aws */
        { PCI_DEVICE(0x1d0f, 0xf000), },
        { PCI_DEVICE(0x1d0f, 0xf001), },
```

- You can execute the `create_design.tcl` script in the Tcl Console of Vivado Design Suite to create a project and test the overall process.
- After the entire project is created, you can use Vivado Design Suite to open the .dcp file and view project information such as information about resources, timing, and wiring.

Instructions on the RTL design on an FPGA-accelerated F3 instance

For information about the instructions on the RTL design on an FPGA-accelerated F3 instance, see Use the RTL design on an f3 instance.

1.3. Use the RTL design on an f3 instance

This topic describes how to implement the Register Transfer Level (RTL) design on an f3 instance.

Prerequisites

- An f3 instance is created and can access the Internet. For more information, see Create an f3 instance.
- A rule is added to the security group to which the f3 instance belongs to allow access over SSH port 22. For more information, see Add a security group rule.
- The ID of the f3 instance is obtained on the Instances page of the Elastic Compute Service (ECS) console.
- An Object Storage Service (OSS) bucket dedicated to FPGA as a Service (FaaS) is created in the China (Shanghai) region. For more information, see Create buckets.

  > **Note** The FaaS administrator account has read and write permissions on the bucket. We recommend that you only store information that is related to FaaS.

-

Context

Before you perform the operations, take note of the following items:

- All operations described in this topic must be performed by a single account within the same region.
- We recommend that you perform operations on an f3 instance as a Resource Access Management (RAM) user. We recommend that you grant only the RAM user permissions based on the principle of least privilege, such as permissions to access DCP/xclbin files in OSS buckets, upload the Vivado compilation log, and manage specified ECS instances. You must also specify the RAM role AliyunFAASDefaultRole. By default, the role is used by FaaS to access resources that are hosted in other cloud services. The AliyunFAASRolePolicy policy of the role includes permissions on Key Management Service (KMS) for you to encrypt IP addresses.

Procedure

1. Connect to an f3 instance.

   > **Note** The compilation process can take two to three hours to complete. We recommend that you use nohup or Virtual Network Computing (VNC) to connect to the instance to avoid unexpected disconnections.

2. Download and decompress the RTL design.
3. Set up the environment.
   - If the driver is `xdma`, run the following command to set up the environment:

     ```
     source /root/xbinst_oem/F3_env_setup.sh xdma #Run this command each time you open a new terminal window.
     ```

   - If the driver is `xocl`, run the following command to set up the environment:

     ```
     source /root/xbinst_oem/F3_env_setup.sh xocl #Run this command each time you open a new terminal window.
     ```

> **Note** The environment set-up process involves installing the xdma or xocl driver, setting the vivado environment variable, checking the vivado license, checking the aliyun-f3 sdaccel platform, configuring 2018.2 runtime, and checking the faascmd version.

4. Specify an OSS bucket.

```
faascmd config --id=<hereIsYourSecretId> --key=<hereIsYourSecretKey> #Replace <hereIsYo
urSecretId> with the AccessKey ID of the RAM user, and replace <hereIsYourSecretKey> wi
th the AccessKey secret of the RAM user.
faascmd auth --bucket=hereIsYourBucket # Replace <hereIsYourBucket> with the name of th
e OSS bucket that you created.
```

5. Run the following command to compile the RTL project:

```
cd <Directory where the decompressed RTL design resides>/hw/ # Access the hw directory
in which the decompressed RTL design resides.
sh compiling.sh
```

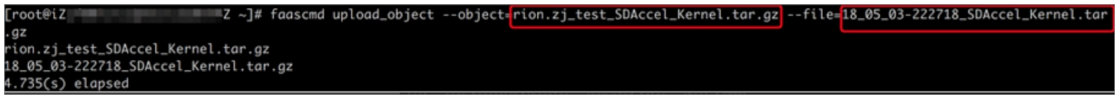
> **Note** The compilation process takes about two to three hours to complete.

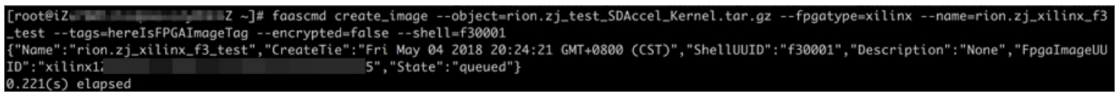
6. Upload the netlist files and download the FPGA image.

   You can use faascmd to upload the netlist files and download the FPGA image. For more information about how to use the faascmd command, see Use faascmd.

   i. Run the following commands in sequence to upload the package to your OSS bucket, and then upload the GBS file from your OSS bucket to the OSS bucket in the FaaS administrative unit:

```
faascmd upload_object --object=bit.tar.gz --file=bit.tar.gz
faascmd create_image --object=bit.tar.gz --fpgatype=xilinx --name=<hereIsFPGAImageN
ame> --tags=<hereIsFPGAImageTag> --encrypted=false --shell=<hereIsShellVersionOfFPG
A>
```



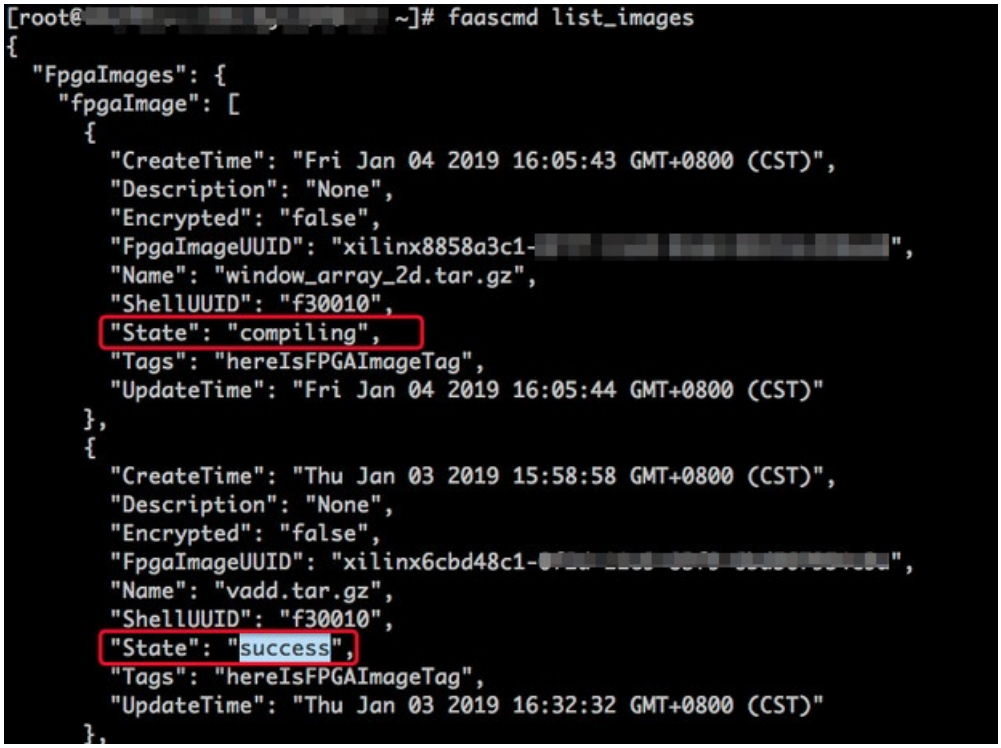


ii. Run the following command to check whether the FPGA image can be downloaded:

```
faascmd list_images
```

The command output is returned:

- If `"State" : "compiling"` is displayed, the FPGA image is being compiled.
- If `"State" : "success"` is displayed, the FPGA image can be downloaded. You must find and record the FpgaImageUUID value.

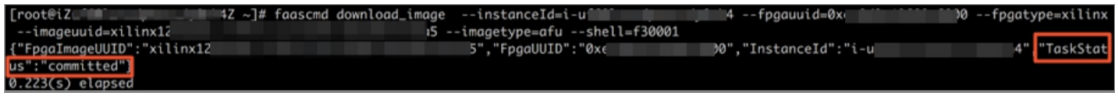


iii. Run the following command. Find and record the FpgaUUID value in the command output.

```
faascmd list_instances --instanceId=<hereIsYourInstanceId> # Replace <hereIsYourInstanceId> with the ID of the f3 instance.
```

iv. Run the following command to download the FPGA image:

```
#Replace <hereIsYourInstanceId> with the ID the f3 instance. Replace <hereIsFpgaUUID> with the FpgaUUID value that you recorded. Replace <hereIsImageUUID> with the FpgaImageUUID value that you recorded.
faascmd download_image --instanceId=<hereIsYourInstanceId> --fpgauuid=<hereIsFpgaUUID> --fpgatype=xilinx --imageuuid=<hereIsImageUUID> --imagetype=afu --shell=<hereIsShellVersionOfFpga>
```



v. Run the following command to check whether the image is downloaded:

```
# Replace <hereIsFpgaUUID> with the FpgaUUID value that you recorded. Replace <hereIsYourInstanceId> with the ID of the f3 instance.
faascmd fpga_status --fpgauuid=<hereIsFpgaUUID> --instanceId=<hereIsYourInstanceId>
```

The following figure shows the command output. If the FpgaImageUUID value in the command output is the same as the FpgaImageUUID value that you recorded, and if `"TaskStatus":"valid"` is displayed, the image is downloaded.

FAQ

Question 1: How do I view the details of errors that occur during the image upload process?

If an exception occurs in your project when you upload an image, you can run the command to view the compilation log file. For example, if an error message is returned on the cloud compilation server, you can view the compilation log file: `sh /root/xbinst_oem/tool/faas_checklog.sh <bit.tar.gz the file name of the compressed package that is uploaded>` .

Question 2: How do I reload an image?

You can perform the following operations to reload an image.

1. Uninstall the driver.
   - If you installed the `xdma` driver, run the `sudo rmmod xdma` command in the instance to uninstall the driver.
   - If you installed the `xocl` driver, run the `sudo rmmod xocl` command in the instance to uninstall the driver.
2. Download the image.

   ```
   faascmd download_image --instanceId=hereIsYourInstanceId --fpgauuid=hereIsFpgaUUID --fpgatype=xilinx --imageuuid=hereIsImageUUID --imagetype=afu --shell=hereIsShellVersionOfFpga
   ```

3. Install the driver.
   - Run the following commands to install the `xdma` driver:

     ```
     sudo depmod
     sudo modprobe xdma
     ```

   - Run the following commands to install the `xocl` driver:

     ```
     sudo depmod
     sudo modprobe xocl
     ```

2.Best practices for using OpenCL on an FPGA-accelerated instances

2.1. Use OpenCL on an f1 instance

This topic describes how to use Open Computing Language (OpenCL) on an f1 instance to create an image and download the image to a Field Programmable Gate Array (FPGA).

Prerequisites

-
-
-
-
-
-
-

Context

Before you perform the operations, take note of the following items:

Procedure

Perform the following operations to use the OpenCL example on an f1 instance to create an image and download the image to an FPGA:

1. Step 1. Connect to an f1 instance
2. Step 2: Install the basic environment
3. Step 3: Copy the OpenCL example
4. Step 4: Upload the configuration file
5. Step 5: Download the image to the f1 instance
6. Step 6: Download the FPGA image to an FPGA

Step 1. Connect to an f1 instance

For more information, see Connect to a Linux instance by using a username and password.

Step 2: Install the basic environment

Run the following script to install the basic environment:

```
source /opt/dcp1_1/script/f1_env_set.sh
```

Step 3: Copy the OpenCL example

1. Run the following commands in sequence to create and switch to the */opt/tmp* directory:

```
mkdir -p /opt/tmp
```

```
cd /opt/tmp
```

Make sure that you are in the */opt/tmp* directory.

2. Run the following command to copy the OpenCL example to the current directory:

```
cp /opt/dcp1_1/opencl/exm_opencl_vector_add_x64_linux.tgz ./
```

3. Run the following commands in sequence to go to the *vector_add* directory for compilation:

```
cd vector_add
```

```
aoc -v -g -report ./device/vector_add.cl
```

The compilation process may take a few hours. You can start another session, and run the **top** command to monitor system resource usage and view the compilation status.

Step 4: Upload the configuration file

1. Run the following commands in sequence to initialize the faascmd tool:

```
#Configure the environment variable.
export PATH=$PATH:/opt/dcp1_1/script/
```

```
#Grant execute permissions to the faascmd tool.
chmod +x /opt/dcp1_1/script/faascmd
```

```
#Replace <hereIsYourSecretId> in the command with your AccessKey ID. Replace <hereIsYou
rSecretKey> with your AccessKey secret.
faascmd config --id=<hereIsYourSecretId> --key=<hereIsYourSecretKey>
```

```
#Replace <hereIsYourBucket> in the command with your bucket name.
faascmd auth --bucket=<hereIsYourBucket>
```

2. Run the following commands in sequence to access the *vector_add/output_files* directory and upload the configuration file:

```
cd vector_add/output_files
```

Make sure that you are in the */opt/tmp/vector_add/vector_add/output_files* directory.

```
faascmd upload_object --object=afu_fit.gbs --file=afu_fit.gbs
```

3. Run the following command to use the GBS file to create an FPGA image:

```
#Replace <hereIsYourImageName> in the command with your image name. Replace <hereIsYour
ImageTag> with your image tag.
faascmd create_image --object=dma_afu.gbs --fpgatype=intel --name=<hereIsYourImageName>
--tags=<hereIsYourImageTag> --encrypted=false --shell=V1.1
```

4. Run the following command to check whether the image is created:

```
faascmd list_images
```

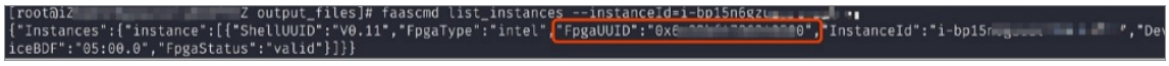
If `"State":"success"` is displayed in the command output, the image is created. Record the FpgaImageUUID value in the command output for subsequent use.

Step 5: Download the image to the f1 instance

1. Run the following command to obtain the ID of your FPGA instance:

```
#Replace <hereIsYourInstanceId> in the command with the ID of your FPGA instance.
faascmd list_instances --instanceId=<hereIsYourInstanceId>
```

The following figure shows the command output. Record the FpgaUUID value.



2. Run the following command to download the image to the f1 instance.

```
#Replace <hereIsYourInstanceID> in the command with the instance ID that you recorded.
Replace <hereIsFpgaUUID> with the value of FpgaUUID that you recorded. Replace <hereIsI
mageUUID> with the value of FpgaImageUUID that you recorded.
faascmd download_image  --instanceId=<hereIsYourInstanceID> --fpgauuid=<hereIsFpgaUUID>
--fpgatype=intel --imageuuid=<hereIsImageUUID> --imagetype=afu --shell=V0.11
```

3. Run the following command to check whether the image is downloaded:

```
# Replace <hereIsYourInstanceID> in the command with the instance ID that you recorded.
Replace <hereIsFpgaUUID> with the value of FpgaUUID that you recorded.
faascmd fpga_status --fpgauuid=<hereIsFpgaUUID> --instanceId=<hereIsYourInstanceID>
```

If `"TaskStatus":"operating"` is displayed in the command output, the image is downloaded.

Step 6: Download the FPGA image to an FPGA

1. Open the environment window in Step 2. If the window is closed, perform the operations in Step 2 again.
2. Run the following command to configure the runtime environment for OpenCL:

```
sh /opt/dcp1_1/opencl/opencl_bsp/linux64/libexec/setup_permissions.sh
```

3. Run the following command to go back to the parent directory:

```
cd ../..
```

Make sure that you are in the */opt/tmp/vector_add* directory.

4. Run the following commands for compilation:

```
make
# Show the environment configuration.
export CL_CONTEXT_COMPILER_MODE_ALTERA=3
cp vector_add.aocx ./bin/vector_add.aocx
cd bin
host vector_add.aocx
```

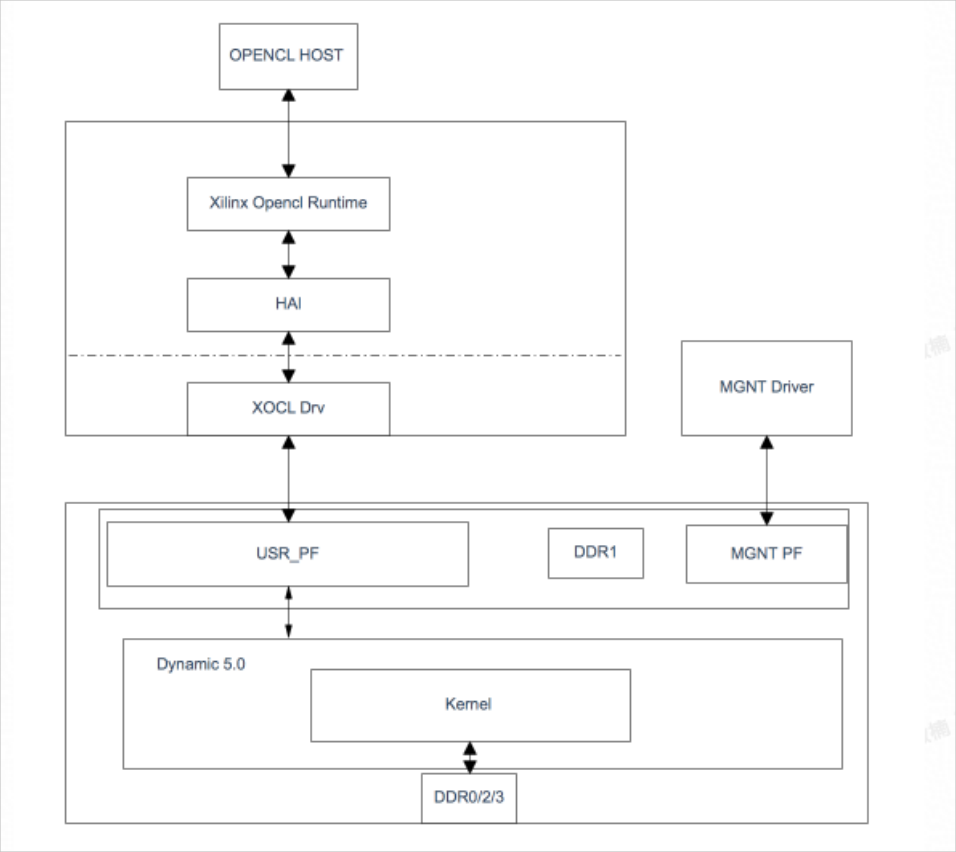
If the following output is displayed, the environment is configured. Note that the last line must be `Verification: PASS` .

```
[root@iZbpXXXXXZ bin]# ./host vector_add.aocx
Matrix sizes:
  A: 2048 x 1024
  B: 1024 x 1024
  C: 2048 x 1024
Initializing OpenCL
Platform: Intel(R) FPGA SDK for OpenCL(TM)
Using 1 device(s)
  skx_fpga_dcp_ddr : SKX DCP FPGA OpenCL BSP (acl0)
Using AOCX: vector_add.aocx
Generating input matrices
Launching for device 0 (global size: 1024, 2048)
Time: 40.415 ms
Kernel time (device 0): 40.355 ms
Throughput: 106.27 GFLOPS
Computing reference output
Verifying
Verification: PASS
```

2.2. Overview of the FaaS f3 SDAccel development environment

The FPGA as a Service (FaaS) f3 SDAccel development environment is based on Xilinx SDAccel dynamic 5.0. You can develop and apply the FaaS f3 SDAccel development environment based on Open Computing Language (OpenCL). This topic describes the SDAccel development environment for f3 instances.

Description of the FaaS f3 SDAccel framework



The following section describes the FaaS SDAccel solution:

- Xilinx OpenCL Runtime: the Xilinx OpenCL runtime that can be used to show OpenCL API operations
- HAL: the Hardware Abstraction Layer (HAL) in which the OpenCL runtime and kernel drivers are adapted and the global memory address is managed
- XOCL Drv: the Xilinx xocl kernel driver
- Host Mgnt Drv: the management driver that runs on the host to load the Field Programmable Gate Array (FPGA) kernel
- User PF: the user physical function (PF) interface that is deployed to a virtual machine to provide users with FPGA access channels
- Mgmt. PF: the management PF interface that serves as the channel for the host to access FPGAs
- kernel: the OpenCL kernel module

Description of the FaaS f3 SDAccel development modules

Development module	Description
Standard OpenCL framework	For more information, see Standard OpenCL framework.
Host code development	Xilinx UG1023
Kernel code development	Xilinx UG1207

User guide for FaaS f3 SDAccel

For more information about the official user guide of FaaS f3 SDAccel, see Use OpenCL on an f3 instance.

2.3. Use OpenCL on an f3 instance

This topic describes how to use Open Computing Language (OpenCL) on an f3 instance to create an image and load the image to a field programmable gate array (FPGA).

Prerequisites

- An f3 instance is created and assigned a public IP address.

  For more information, see Create an f3 instance.

  > **Note** Only images shared by Alibaba Cloud can be used on f3 instances.

- A rule is added to the security group to which the f3 instance belongs to allow access over SSH port 22.
- The ID of the f3 instance is obtained from the Instances page of the Elastic Compute Service (ECS) console.
- An Object Storage Service (OSS) bucket dedicated to FPGA as a Service (FaaS) is created.

  The bucket and the f3 instance belong to the same account and are deployed in the same region. For information about how to create a bucket, see Create buckets.

-

Context

Before you perform the operations, take note of the following items:

- All operations described in this topic can be performed only by one account in the same region.
- We recommend that you manage the f3 instance as a Resource Access Management (RAM) user. You must create a role for the RAM user and grant the role temporary permissions to access the specified OSS bucket.
- The operations and commands described in this topic are based on version 2018.2 of the SDAccel development environment. If you use the SDAccel development environment of other versions, the operations and commands vary.

Procedure

Perform the following operations to use OpenCL on the f3 instance to create an image and load the image to an FPGA.

1. Step 1: Set up the environment
2. Step 2: Compile a binary file
3. Step 3: Check the packaging script
4. Step 4: Create an image
5. Step 5: Download the image
6. Step 6: Run the host program

Step 1: Set up the environment

1. Remotely connect to an f3 instance.

> **Note** The compilation process can take several hours. We recommend that you log on by using Screen or nohup to avoid a forced logout due to an SSH timeout.

2. Run the following command to install Screen:

```
yum install screen -y
```

3. Run the following command to start Screen:

```
screen -S f3opencl
```

4. Run the following command to set up the environment:

```
source /root/xbinst_oem/F3_env_setup.sh xocl  #Run this command each time you open a new terminal window.
```

> **Note**
> - To set up the environment, you need to install the xocl driver, configure the environment variables of Vivado, check the license of Vivado, detect the aliyun-f3 SDAccel development environment, configure the 2018.2 runtime, and check the version of faascmd.
> - If you want to run a Vivado emulator, do not run the preceding commands to set up the environment. You need to separately configure the environment variable only for Vivado.
> - We recommend that you use Makefile for emulation.

5. Go to Xilinx, and download and install the y2k22_patch-1.2.zip patch.

> **Note** This patch is used to fix the 2022 timestamp overflow bug in Xilinx.

Step 2: Compile a binary file

Perform the following operations to compile the vadd and kernel_global_bandwidth binary files:

- Example 1: Compile the vadd binary file
  i. Copy the *example* directory.

```
cp -rf /opt/Xilinx/SDx/2018.2/examples ./
```

  ii. Go to the *vadd* directory.

```
cd examples/vadd/
```

  iii. Run the **cat sdaccel.mk | grep "XDEVICE="** command to check the value of XDEVICE. Make sure that the value is `xilinx_aliyun-f3_dynamic_5_0`.
  iv. Perform the following operations to modify the *common.mk* file:
    a. Run the **vim ../common/common.mk** command to open the file.

b. At the end of code line 61, add the compilation parameter --xp param:compiler.acceleratorBinaryContent=dcp. The parameter can be added in code line 60, 61, or 62. The following section shows the modified code:

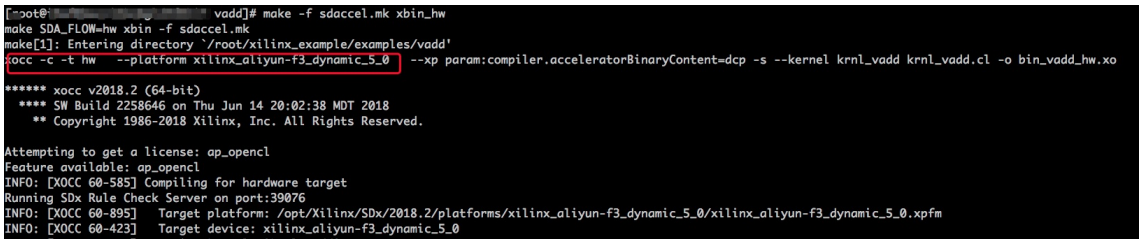
```
CLCC_OPT += $(CLCC_OPT_LEVEL) ${DEVICE_REPO_OPT} --platform ${XDEVICE} ${KERNEL_D
EFS} ${KERNEL_INCS} --xp param:compiler.acceleratorBinaryContent=dcp
```

> **Note** You must add the compilation parameter --xp param:compiler.acceleratorBinaryContent=dcp so that Xilinx® OpenCL™ Compiler (xocc) can generate a DCP file after the placement and routing are completed. The file cannot be a bit file. Then, you can submit the DCP file to the compilation server.

v. Run the following command to compile the program:

```
make -f sdaccel.mk xbin_hw
```

If the information displayed is similar to the following example, the binary file is being compiled. The compilation process can take several hours.



- Example 2: Compile the kernel_global_bandwidth binary file

i. Run the following commands in sequence to clone `xilinx 2018.2 example`:

```
git clone https://github.com/Xilinx/SDAccel_Examples.git
```

```
cd SDAccel_Examples/
```

```
git checkout 2018.2
```

> **Note** The Git branch version must be 2018.2.

ii. Run the **cd getting_started/kernel_to_gmem/kernel_global_bandwidth/** command to access the directory.

iii. Perform the following operations to modify the *Makefile* file:

a. Run the **vim Makefile** command to open the file.

b. Set `DEVICES` to xilinx_aliyun-f3_dynamic_5_0.

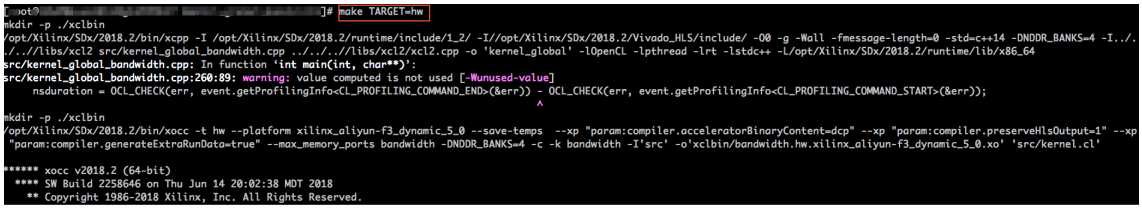
c. In code line 33, add the compilation parameter --xp param:compiler.acceleratorBinaryContent=dcp. The following section shows the modified code:

```
CLFLAGS +=--xp "param:compiler.acceleratorBinaryContent=dcp" --xp "param:compiler
.preserveHlsOutput=1" --xp "param:compiler.generateExtraRunData=true" --max_memor
y_ports bandwidth -DNDDR_BANKS=$(ddr_banks)
```

iv. Run the following command to compile the program:

```
make TARGET=hw
```

If the information displayed is similar to the following example, the binary file is being compiled. The compilation process can take several hours.



Step 3: Check the packaging script

Run the following command to check whether the packaging script exists:

```
file /root/xbinst_oem/sdaccel_package.sh
```

If `cannot open (No such file or directory)` is displayed in the command output, the script does not exist. You must run the following command to manually download the packaging script:

```
wget http://fpga-tools.oss-cn-shanghai.aliyuncs.com/sdaccel_package.sh
```

Step 4: Create an image

1. Initialize the faascmd tool and set up the OSS environment.
   i. Run the following command to configure the AccessKey pair of the RAM user:

   ```
   #Replace <hereIsYourSecretId> with the AccessKey ID of the RAM user, and <hereIsYou
   rSecretKey> with the AccessKey secret of the RAM user.
   faascmd config --id=<hereIsYourSecretId> --key=<hereIsYourSecretKey>
   ```

   ii. Run the following command to configure the OSS bucket dedicated to FaaS:

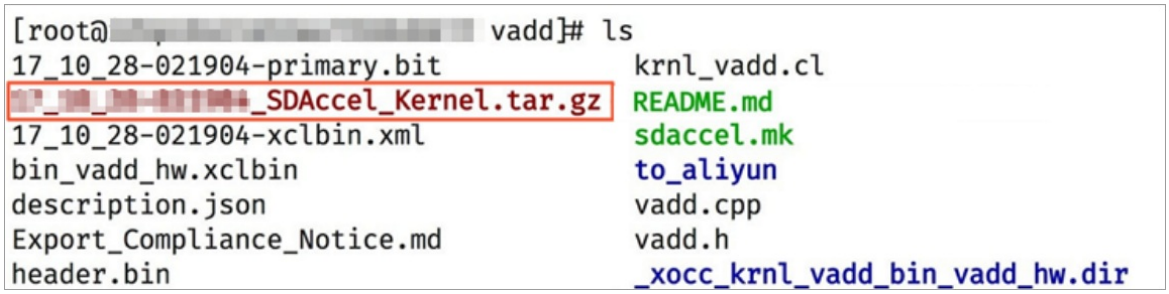
   ```
   #Replace <hereIsYourBucket> with the name of the OSS bucket that you created.
   faascmd auth --bucket=<hereIsYourBucket>
   ```

2. Run the ls command to obtain the file with the suffix `.xclbin`.

3. Run the following command to package the binary file:

   ```
   /root/xbinst_oem/sdaccel_package.sh -xclbin=/opt/Xilinx/SDx/2018.2/examples/vadd/bin_va
   dd_hw.xclbin
   ```

After the binary file is packaged, you can find the packaged file in the same directory, as shown in the following figure.



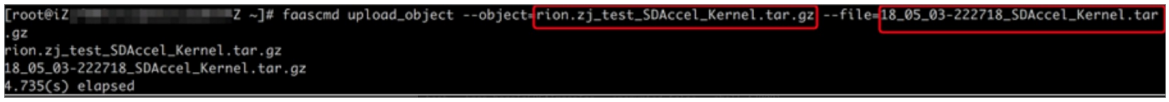
Step 5: Download the image

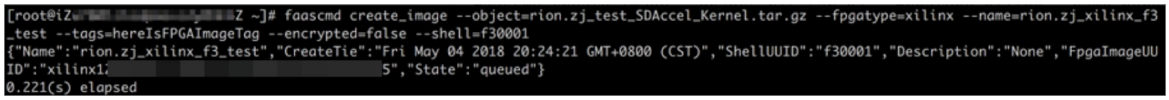
This section describes how to use the faascmd tool to upload a netlist file and download an FPGA image. For information about how to use the faascmd commands, see Use the faascmd tool.

1. Run the following commands in sequence to upload the package to your OSS bucket, and then upload the GBS file from your OSS bucket to the OSS bucket in the FaaS administrative unit:

   ```
   faascmd upload_object --object=bit.tar.gz --file=bit.tar.gz
   ```

   ```
   faascmd create_image --object=bit.tar.gz --fpgatype=xilinx --name=<hereIsFPGAImageName>
   --tags=<hereIsFPGAImageTag> --encrypted=false --shell=<hereIsShellVersionOfFPGA>
   ```

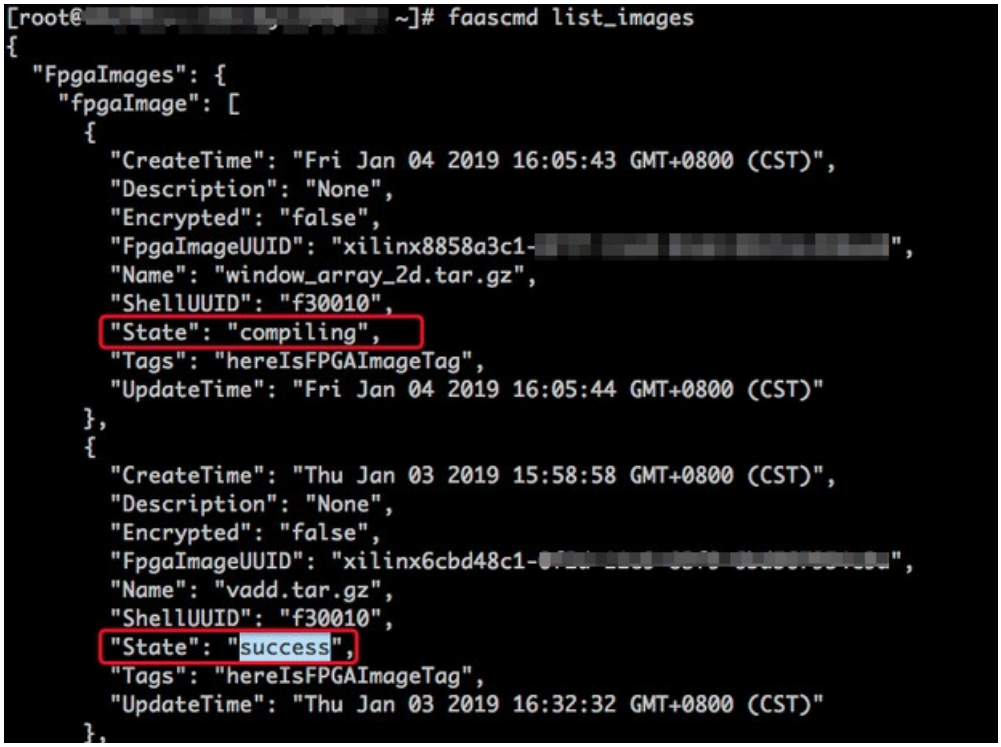
   

2. Run the following command to check whether the FPGA image can be downloaded:

   ```
   faascmd list_images
   ```

   The following section shows the command output:

   - If `"State" : "compiling"` is displayed, the FPGA image is being compiled.
   - If `"State" : "success"` is displayed, the FPGA image can be downloaded. You must find and record the value of FpgaImageUUID.

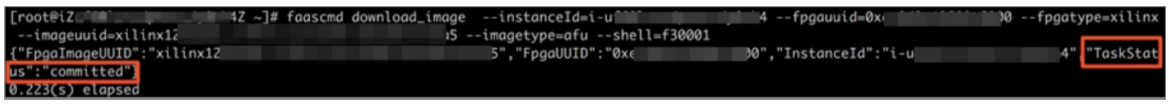


3. Run the following command. Find and record the value of FpgaUUID in the command output.

```
faascmd list_instances --instanceId=<hereIsYourInstanceId> #Replace <hereIsYourInstanceId> with the ID of the f3 instance.
```

4. Run the following command to download the FPGA image:

```
#Replace <hereIsYourInstanceId> with the ID the f3 instance. Replace <hereIsFpgaUUID> with the value of FpgaUUID that you recorded. Replace <hereIsImageUUID> with the value of FpgaImageUUID that you recorded.
faascmd download_image --instanceId=<hereIsYourInstanceId> --fpgauuid=<hereIsFpgaUUID> --fpgatype=xilinx --imageuuid=<hereIsImageUUID> --imagetype=afu --shell=<hereIsShellVersionOfFpga>
```



5. Run the following command to check whether the image is downloaded:

```
faascmd fpga_status --fpgauuid=<hereIsFpgaUUID> --instanceId=<hereIsYourInstanceId> #Replace <hereIsFpgaUUID> with the value of FpgaUUID that you recorded earlier. Replace <hereIsYourInstanceId> with the ID of the f3 instance.
```

The following figure shows the command output. If the value of FpgaImageUUID in the command output is the same as the FpgaImageUUID value that you recorded and `"TaskStatus":"valid"` is displayed, the image is downloaded.

Step 6: Run the host program

1. Run the following command to set up the environment:

```
source /root/xbinst_oem/F3_env_setup.sh xocl  #Run this command each time you open a new terminal window.
```

2. Configure the *sdaccel.ini* file.

   In the directory where the Host binary file is located, run the **vim sdaccel.ini** command to create the *sdaccel.ini* file and enter the following content:

```
[Debug]
profile=true
[Runtime]
runtime_log = "run.log"
hal_log = hal.log
ert=false
kds=false
```

3. Run the host program.
   - For vadd, run the following commands:

```
make -f sdaccel.mk host
```

```
./vadd bin_vadd_hw.xclbin
```

   - For kernel_global_bandwidth, run the following command:

```
./kernel_global
```

   If `Test Passed` is displayed in the command output, the test is passed.

What to do next

The following table describes some common commands that you can run to perform operations on f3 instances.

Operation	Command
View the help documentation	`make -f ./sdaccel.mk help`
Run software emulation	`make -f ./sdaccel.mk run_cpu_em`
Run hardware emulation	`make -f ./sdaccel.mk run_hw_em`
Compile only the host code	`make -f ./sdaccel.mk host`
Compile and generate downloadable files	`make -f sdaccel.mk xbin_hw`
Clean a working directory	`make -f sdaccel.mk clean`
Forcibly clean a working directory	`make -f sdaccel.mk cleanall`

Note

- During emulation, follow the Xilinx emulation process. You do not need to configure the F3_env_setup environment.
- The SDAccel runtime and SDAccel development platform are available in the official f3 images that are provided by Alibaba Cloud. You can also click SDAccel runtime and SDAccel development platform to download the images.

3.Use Vitis 2020.1 on an f3 instance

This topic describes how to use Vitis 2020.1 on an f3 instance to create an image and load the image to a field programmable gate array (FPGA).

Prerequisites

- An f3 instance that meets the following requirements is created. For more information, see Create an f3 instance.
  - You must submit a ticket to obtain the image **FaaS_F30010_VITIS_2020_1**, and select the image when you create the instance.
  - The size of the system disk is equal to or greater than 120 GiB.
  - A public IP address is assigned to the f3 instance.
  - A rule is added to the security group to which the f3 instance belongs to allow access over SSH port 22.
- The ID of the f3 instance is obtained from the Instances page of the Elastic Compute Service (ECS) console.
- An Object Storage Service (OSS) bucket dedicated to FPGA as a Service (FaaS) is created.

  The bucket and the f3 instance belong to the same account and are deployed in the same region. For information about how to create a bucket, see Create buckets.
-

Procedure

Perform the following operations to use Vitis 2020.1 on the f3 instance to create an image and load the image to an FPGA.

1. Step 1: Remotely connect to an f1 instance
2. Step 2: Initialize the software environment
3. Step 3: Create a project
4. Step 4: Use Vitis for emulation
5. Step 5: Create an image
6. Step 6: Verify the result

Step 1: Remotely connect to an f1 instance

A desktop environment and a Virtual Network Console (VNC) server have been configured for the **FaaS_F30010_VITIS_2020_1** image. We recommend that you remotely connect to the f3 instance by using the VNC server. For more information, see Connect to a Linux instance by using a password.

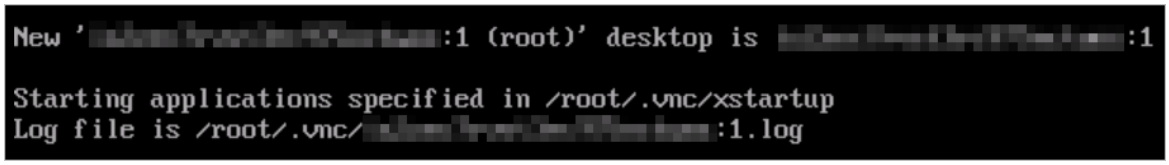
Step 2: Initialize the software environment

Each time you create an f3 instance, you must perform the following operations to initialize the software environment.

1. Run the following command to specify an environment resolution by using the VNC server:

```
vncserver -geometry 2560x1440
```

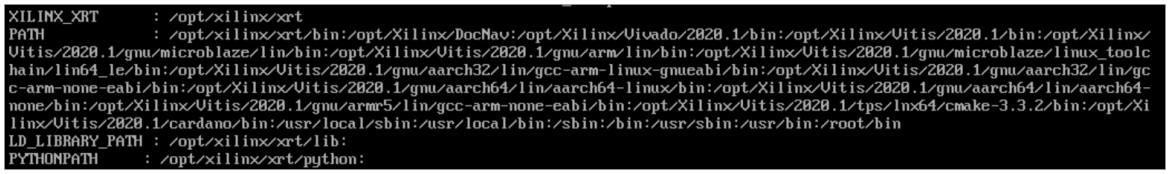
The following information is returned.



2. Run the following command to initialize the software environment that is pre-installed in the FaaS platform and is required when you use Vitis in the **FaaS_F30010_VITIS_2020_1** image:

```
source /root/faasTools/vitis_setup.sh
```

If the following information is returned, the software environment is initialized.

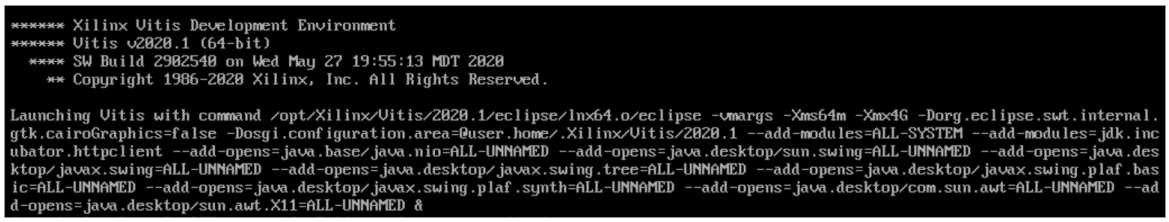


Step 3: Create a project

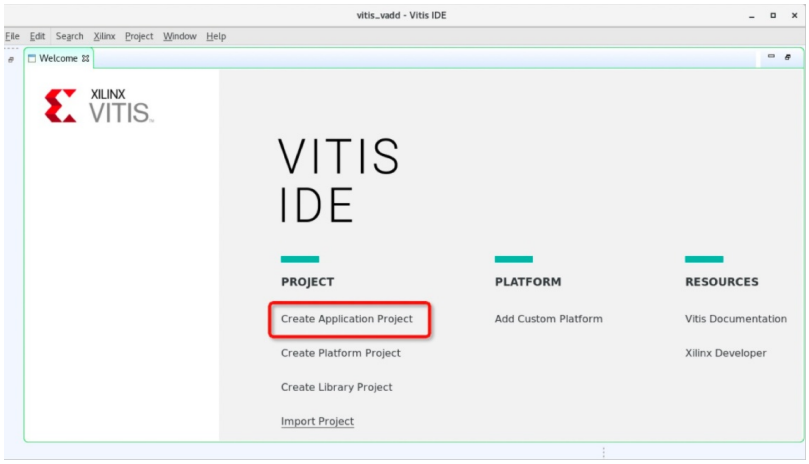
After the environment is ready, you can start the project by running the vitis command, and create a project on the GUI of Vitis.

1. Run the `vitis` command to start the project.

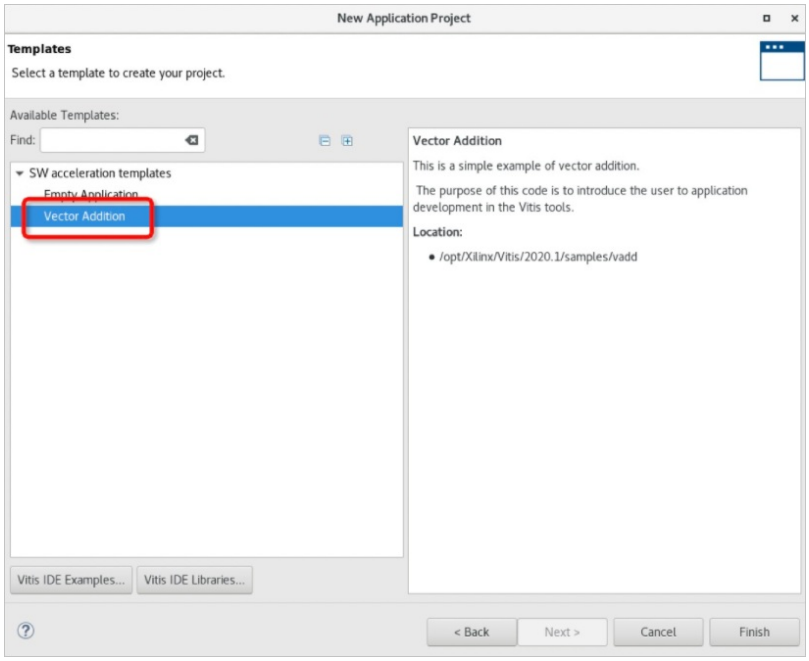
   If the following information is returned, the project is started.



2. After the project is started, perform the following steps to create a project on the GUI of Vitis.
   i. In the **VITIS** window, click **Create Application Project** below **PROJECT**.



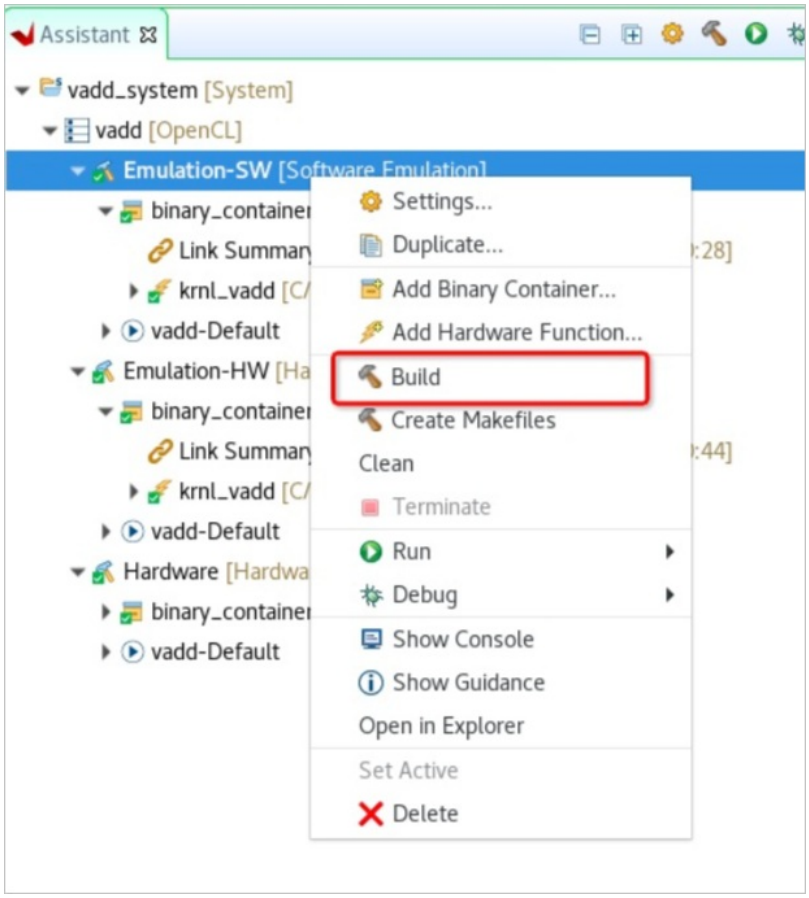
ii. In the **New Application Project** dialog box, choose **SW acceleration templates > Vector Addition** in the left-side section, and then click **Finish** in the lower-right corner.



Step 4: Use Vitis for emulation

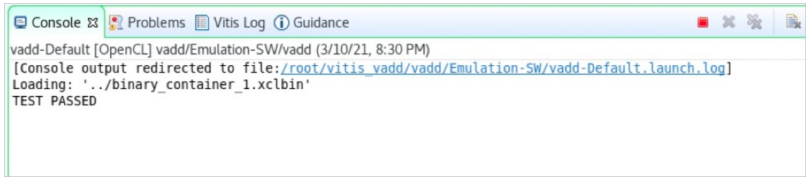
Vitis supports two emulation modes: Emulation-SW and Emulation-HW.

1. On the **Assistant** tab in the GUI of Vitis, choose **vadd_system > vadd > Emulation-SW [Software Emulation]**.
2. Right-click **Emulation-SW [Software Emulation]**, and select **Build** from the shortcut menu.



3. Right-click **Emulation-SW [Software Emulation]**, and choose **Run > Default** to emulate the model of the project that you created.

   The following result is returned.



Step 5: Create an image

By default, the image files that are generated by Vitis include files with the .bit extension. FaaS requires that the files with the .dcp extension be uploaded. Therefore, you must configure the relevant settings before you create an FPGA image.

1. In the upper-right corner of the **Assistant** tab, click the  .
2. In the **Project Settings** dialog box, choose **vadd_system > vadd**.
3. In the **vadd** section on the right side, specify the following content for the **V++ linker options** configuration item. Click **Apply**, and then click **Apply and Close**.

   ```
   --advanced.param compiler.acceleratorBinaryContent=dcp
   ```

4. Return to the **Assistant** tab, and choose **vadd_system > vadd > Hardware [Hardware]**.
5. Right-click **Hardware [Hardware]**, and select **Build** from the shortcut menu to create the image.

It may take several hours to create the image. After the image is created, you can go to the Hardware directory in the project directory and run the `ls` command to view the generated binary file vadd that can be executed by the host and the image file binary_container_1.xclbin. The following sample code shows the command output:

```
[root@iz2zec7rvsxxxxxxx Hardware]# ls
a.xclbin                                guidance.html
binary_container_1.build                guidance.pb
binary_container_1-krnl_vadd-compile.cfg makefile
binary_container_1-link.cfg             package.build
binary_container_1.ltx                  package.cfg
binary_container_1.mdb                  src
binary_container_1.xclbin               vadd
binary_container_1.xclbin.info          vadd_Hardware.build.ui.log
binary_container_1.xclbin.link_summary  v++_package.log
binary_container_1.xclbin.sh            v++.package_summary
common-config.cfg                       xcd.log
```

6. Run the following command to generate a compressed file for uploading the image:

```
[root@iz2zec7rvsxxxxxxx Hardware]# vitis_xclbin_split.sh binary_container_1.xclbin
```

The following command output is displayed:

```
XRT Build Version: 2.6.0 (2020.1)
       Build Date: 2021-03-08 10:50:41
          Hash ID: 80107ebc7376dafc8e1c9f5043c81c6f1dcc9dbb
------------------------------------------------------------------------------
Warning: The option '--output' has not been specified. All operations will
         be done in memory with the exception of the '--dump-section' command.
------------------------------------------------------------------------------
Reading xclbin file into memory.  File: binary_container_1.xclbin
Section: 'BITSTREAM'(0) was successfully written.
Format: RAW
File  : 'faas20210311-092706.dcp'
Leaving xclbinutil.
XRT Build Version: 2.6.0 (2020.1)
       Build Date: 2021-03-08 10:50:41
          Hash ID: 80107ebc7376dafc8e1c9f5043c81c6f1dcc9dbb
------------------------------------------------------------------------------
Warning: The option '--output' has not been specified. All operations will
         be done in memory with the exception of the '--dump-section' command.
------------------------------------------------------------------------------
Reading xclbin file into memory.  File: binary_container_1.xclbin
Section: 'EMBEDDED_METADATA'(2) was successfully written.
Format: RAW
File  : 'faas20210311-092706.xml'
Leaving xclbinutil.
to_aliyun/
to_aliyun/faas20210311-092706.xml
to_aliyun/faas20210311-092706.dcp
Generate Image :Image20210311-092706.tar.gz
```

The preceding command output shows that the **Image20210311-092706.tar.gz** file is finally generated. You can use the file to upload the image. For more information, see Use faasutil.

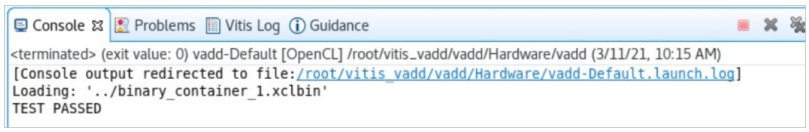
> **Note** The faasutil tool is pre-installed in the FaaS_F30010_VITIS_2020_1 image. You can use the faasutil tool after you run the `source /root/faasTools/vitis_setup.sh` command.

Step 6: Verify the result

After you use the faasutil tool that is pre-installed in the FaaS_F30010_VITIS_2020_1 image to create an image and load the image to the FPGA, you can verify whether the image is loaded to the FPGA by using the CLI or the GUI.

- Use the GUI for verification:
  i. On the **Assistant** tab in the GUI of Vitis, choose **vadd_system > vadd > Hardware [Hardware]**.
  ii. Right-click **Hardware [Hardware]**, and choose **Run > Default**.

  The following result is returned. The result shows that the image is loaded to the FPGA.



- Use the CLI for verification:

  In the Hardware directory of the project directory, run the following command to verify whether the image is loaded to the FPGA:

```
[root@iz2zec7rvsxxxxxxx Hardware]# ./vadd binary_container_1.xclbin
```

  The following result is returned. The result shows that the image is loaded to the FPGA.

```
Loading: 'binary_container_1.xclbin'
TEST PASSED
```