DPA Tools

Table of contents

Introduction

DPA tools are a set of executables that enable the DPA application developer and the system administrator to manage and monitor DPA resources and to debug DPA applications.

DPA Tools

DPACC Compiler

CLI name: `dpacc`

DPACC is a high-level compiler for the DPA processor. It compiles code targeted for the DPA processor into an executable and generates a DPA program.

The DPA program is a host library with interfaces encapsulating the DPA executable. This DPA program can be linked with the host application to generate a host executable where the DPA code is invoked through the FlexIO runtime API.

DPA EU Management Tool

CLI name: `dpaeumgmt`

This tool allows users to manage the DPA's EUs which are the basic resource of the DPA. The tool enables the resource control of EUs to optimize the usage of computation resources of the DPA. Using this tool, users may query, create, and destroy EU partitions and groups , thus ensuring proper EU allocation between devices.

DPA GDB Server Tool

CLI name: `dpa-gdbserver`

The DPA GDB Server tool enables debugging FlexIO DEV programs.

DPA PS Tool

CLI name: `dpa-ps`

This tool allows users to monitor running DPA processes and threads.

DPA Statistic Tool

CLI name: `dpa-statistics`

This tool allows users to monitor and obtain statistics on thread execution per running DPA process and thread.

DOCA DPACC Compiler

Contents:

This document describes DOCA DPACC compiler and instructions about DPA toolchain setup and usage.

Introduction

DPACC is a high-level compiler for the DPA processor which compiles code targeted for the data-path accelerator (DPA) processor into a device executable and generates a DPA program.

The DPA program is a host library with interfaces encapsulating the device executable. This DPA program is linked with the host application to generate a host executable. The host executable can invoke the DPA code through FlexIO runtime API.

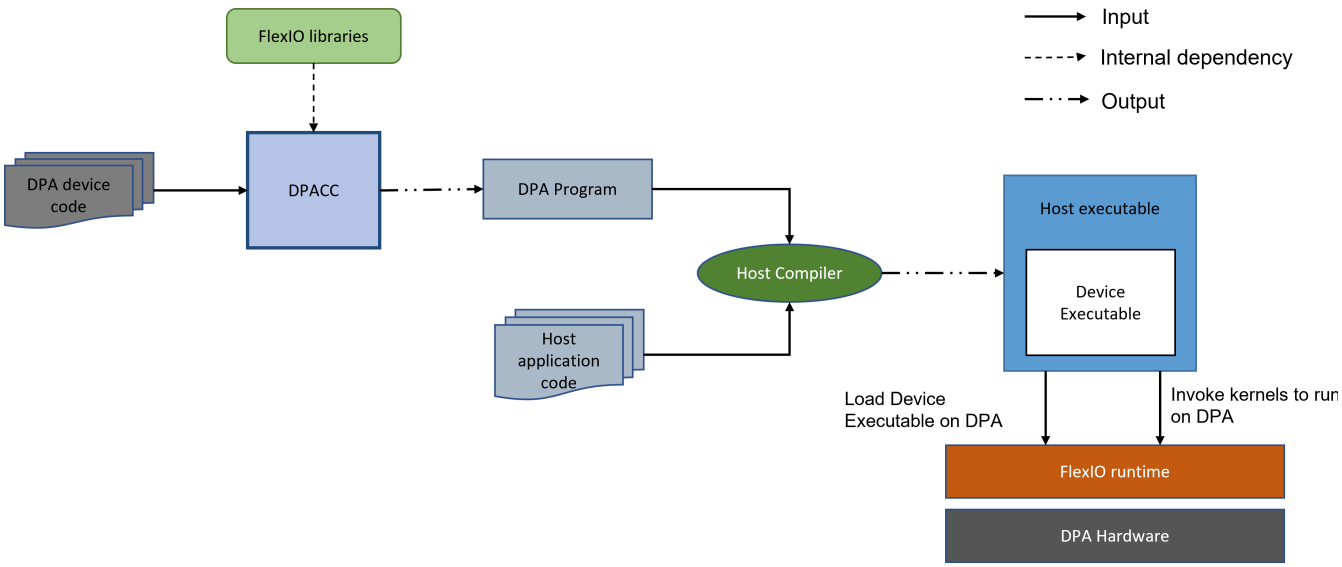
DPACC uses DPA compiler (`dpa-clang`) to compile code targeted for DPA. dpa-clang is part of the DPA toolchain package which is an LLVM-based cross-compiling bare-metal toolchain. It provides Clang compiler, LLD linker targeting DPA architecture, and other utilities.

Glossary

Term	Definition
Device	DPA as present on the BlueField DPU
Host	CPU that launches the device code to run on the DPA
Device function	Any C function that runs on the DPA device
DPA global function	Device function that is the point of entry when offloading any work on DPA
Host compiler	Compiler used to compile the code targeting the host CPU
Device compiler	Compiler used to compile code targeting the DPA

Term	Definition
Fatbinary	File that contains code for multiple target DPA architectures
DPA program	Host library that encapsulates the DPA device executable (`.elf`) and host stubs which are used to access the device executable

Offloading Work on DPA



To invoke a DPA function from host, the following things are required:

- DPA device code – C programs, targeted to run on the DPA. DPA device code may contain one or more entry functions.
- Host application code – the corresponding host application. Please refer to DPA Subsystem for more details
- Runtime – FlexIO or DOCA DPA library provides the runtime

The generated DPA program, when linked with a host application results in a host executable which also contains the device executable. The host application oversees loading the device executable on the device.

DPACC Predefined Macros

DPACC predefines the following macros:

`__DPA__`	Defined when compiling device code file
`__NV_DPA`	Defined to the target DPA hardware identifier macros See Architecture Macros for more details.
`__DPA_MAJOR__`	Defined to the major version number of DPACC
`__DPA_MINOR__`	Defined to the minor version number of DPACC
`__DPA_PATCH__`	Defined to the patch version number of DPACC

Writing DPA Applications

DPA device code is a C code with some restrictions and special definitions.

FlexIO or DOCA-DPA APIs provide interfaces to DPA.

Language Support

The DPA is programmed using a subset of the C11 language standard. The compiler documents any constructs that are not available. Language constructs, where available, retain their standard definitions.

Restrictions on DPA Code

- Use of C thread local storage is not allowed for any variables
- Identifiers with `_dpacc/__dpacc` prefix are reserved by the compiler. Use of such identifiers may result in an error or undefined behavior
- DPA processor does not have native floating-point support; use of floating point operations is disabled

DPA RPC Functions

A remote procedure call function is a synchronous call that triggers work in DPA and waits for its completion. These functions return a type `uint64_t` value. They are annotated

with a `__dpa_rpc__` attribute.

DPA Global Functions

A DPA global function is an event handler device function referenced from the host code. These functions do not return anything. They are annotated with a `__dpa_global__` attribute.

For more information, refer to the *DPA Subsystem*.

Characteristics of Annotated Functions

- Global functions must have `void` return type and RPC functions must have `uint64_t` return type
- Annotated functions cannot accept C pointers and arrays as arguments (e.g., `void my_global (int *ptr, int arr[])`)
- Annotated functions cannot accept a variable number of arguments
- Inline specifier is not allowed on annotated functions

Handling User-defined Data Types

User-defined data types, when used as global function arguments, require special handling. They must be annotated with a `__dpa_global__` attribute.

If the user-defined data type is `typedef`'d, the `typedef` statement must be annotated with a `__dpa_global__` attribute along the data type itself.

Characteristics of Annotated Types

- They must have a copy of the definition in all translation units where they are used as global function arguments

- They cannot have pointers, variable length arrays, and flexible arrays as members
- Fixed-size arrays as C structure members are supported
- These characteristics apply recursively to any user-defined/`typedef`'d types that are members of an annotated type

DPACC processes all annotated functions along with annotated types and generates host and device interfaces to facilitate the function launch.

DPA Intrinsics

DPA features such as fences and processor-specific instructions are exposed via intrinsics by the DPA compiler. All intrinsics defined in the header file `dpaintrin.h` are guarded by the `DPA_INTRIN_VERSION_USED` macro. The current `DPA_INTRIN_VERSION` is `1.3`.

Example:

```
#define DPA_INTRIN_VERSION_USED (DPA_INTRIN_VERSION(1, 3))
#include <dpaintrin.h>
…
__dpa_thread_writeback_window();   // Fence for write barrier
```

For more information, please refer to *DPA Subsystem*.

Prerequisites

Package	Instructions
Host compiler	Compiler specified through `hostcc` option. Both `gcc` and `clang` are supported.

Package	Instructions
	Note Minimum supported version for clang as hostcc is `clang 3.8.0`.
Device compiler	The default device compiler is the "DPA compiler". Installing the DPACC package also installs the DPA compiler binaries `dpa-clang`, `dpa-ar`, `dpa-nm` and `dpa-objdump`. **Note** `dpa-clang` is the only supported device compiler.
FlexIO SDK and C library	Available as part of the DOCA software package. DPA toolchain does not provide C library and corresponding headers. Users are expected to use the C library for DPA from the FlexIO SDK.

Description

DPACC Inputs and Outputs

DPACC can produce DPA programs in a single command by accepting all source files as input. DPACC also offers the flexibility of producing DPA object files or libraries from input files.

DPA object files contain both host stub objects (DPACC-generated interfaces) and device objects. These DPA object files can later be given to DPACC as input to produce the DPA library.

Phase	Option Name	Default Output File Name
Compile input device code files to DPA object files	`--compile` or `-c`	`.dpa.o` appended to the name of each input source file

Phase	Option Name	Default Output File Name
Compile and link the input device code files/DPA object files, and produce a DPA program	No specific option	No default name, output file name must be specified
Compile and build DPA library from input device code files/DPA object files	`--gen-libs` or `-gen-libs`	No default name, output library name must be specified

DPACC can accept the following file types as input:

Input File Extension	File Type	Description
`.c`	C source file	DPA device code
`.dpa.o`	DPA object file	Object file generated by DPACC, containing both host and device objects
`.a`	DPA object archive	An archive of DPA object files. User can generate this archive from DPACC-generated DPA objects.

Based on the mode of operations, DPACC can generate the following output files:

Output File Type	Input Files
DPA object file	C source files
DPA program	C source files, DPA object files, and/or DPA object archives
DPA library (DPA host library and DPA device library)	C source files, DPA object files, and/or DPA object archives

The following provides the commands to generate different kinds of supported output file types for each input file type:

Input	Output	DPACC Command
C source file	DPA program	dpacc -hostcc=<cc> -mcpu=<targets> in.c -o libprog.a
	DPA object	dpacc -hostcc=<cc> -mcpu=<targets> in.c -c

Input	Output	DPACC Command
	DPA library	dpacc -hostcc=<cc> -mcpu=<targets> in.c -o lib<name> -gen-libs
DPA object	DPA program	dpacc -hostcc=<cc> -mcpu=<targets> in.dpa.o -o libprog.a
	DPA library	dpacc -hostcc=<cc> -mcpu=<targets> in.dpa.o -o lib<name> -gen-libs
DPA object archive	DPA program	dpacc -hostcc=<cc> -mcpu=<targets> in.a -o libprog.a
	DPA library	dpacc -hostcc=<cc> -mcpu=<targets> in.a -o lib<name> -gen-libs

DPA Program

DPACC produces a DPA program in compile-and-link mode. A DPA program is a host library which contains:

- DPACC-generated host stubs which registering the DPA application and facilitate invoking an entry-point function from the host application
- Device executable, generated by DPACC by compiling and linking input DPA device code

DPA program library must be linked with the host application that contains appropriate runtime APIs to load the device executable onto the device.

A DPA Program can contain device executables for multiple targets. All the target-specific device executables are encapsulated in a special fatbinary container format and this container is embedded as a section into the host object which is present in the host library.

DPA Object

DPACC produces DPA object files in compile-only mode. A DPA object is a host object file which has a similar layout to the DPA-Program where it contains DPACC-generated host stubs and device object which is generated by compiling input device code.

A DPA Object can contain device objects for multiple targets similar to a DPA-Program. All the target-specific device objects are encapsulated in a special fatbinary container format and this container is embedded as a section into the host object.

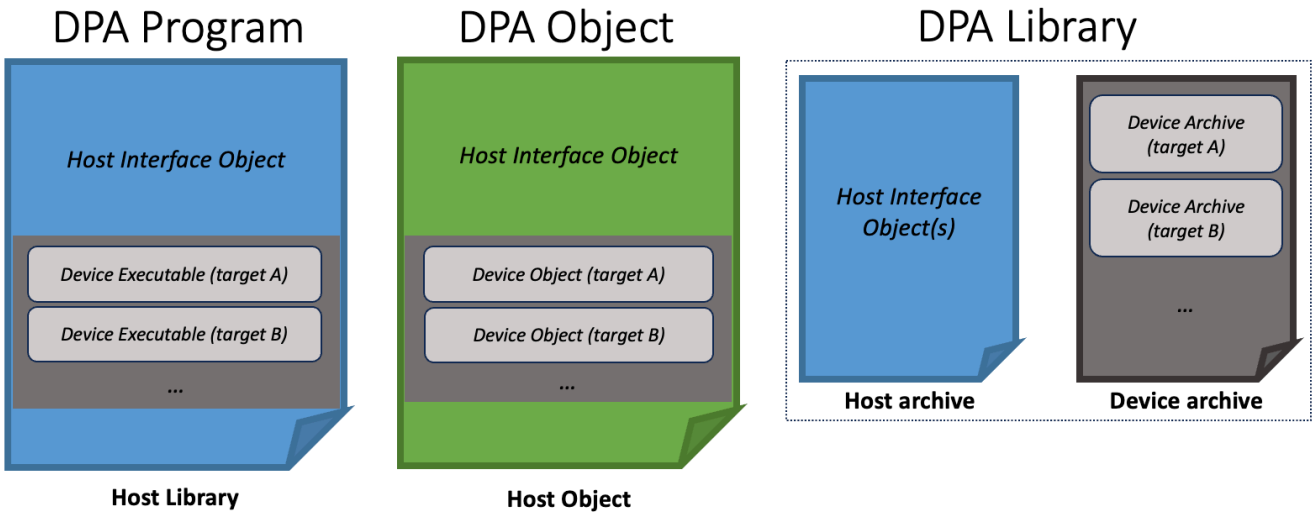
DPA Library

A DPA library is a collection of two individual libraries:

- DPA host library – contains host interface objects corresponding to the device objects in DPA device library
- DPA device library – contains device objects generated by compiling the input device code files

The DPA device library is consumed by DPACC during DPA-program generation and the DPA host library can optionally be linked with other host code and be distributed as the host library. Both libraries are generated as static archives.

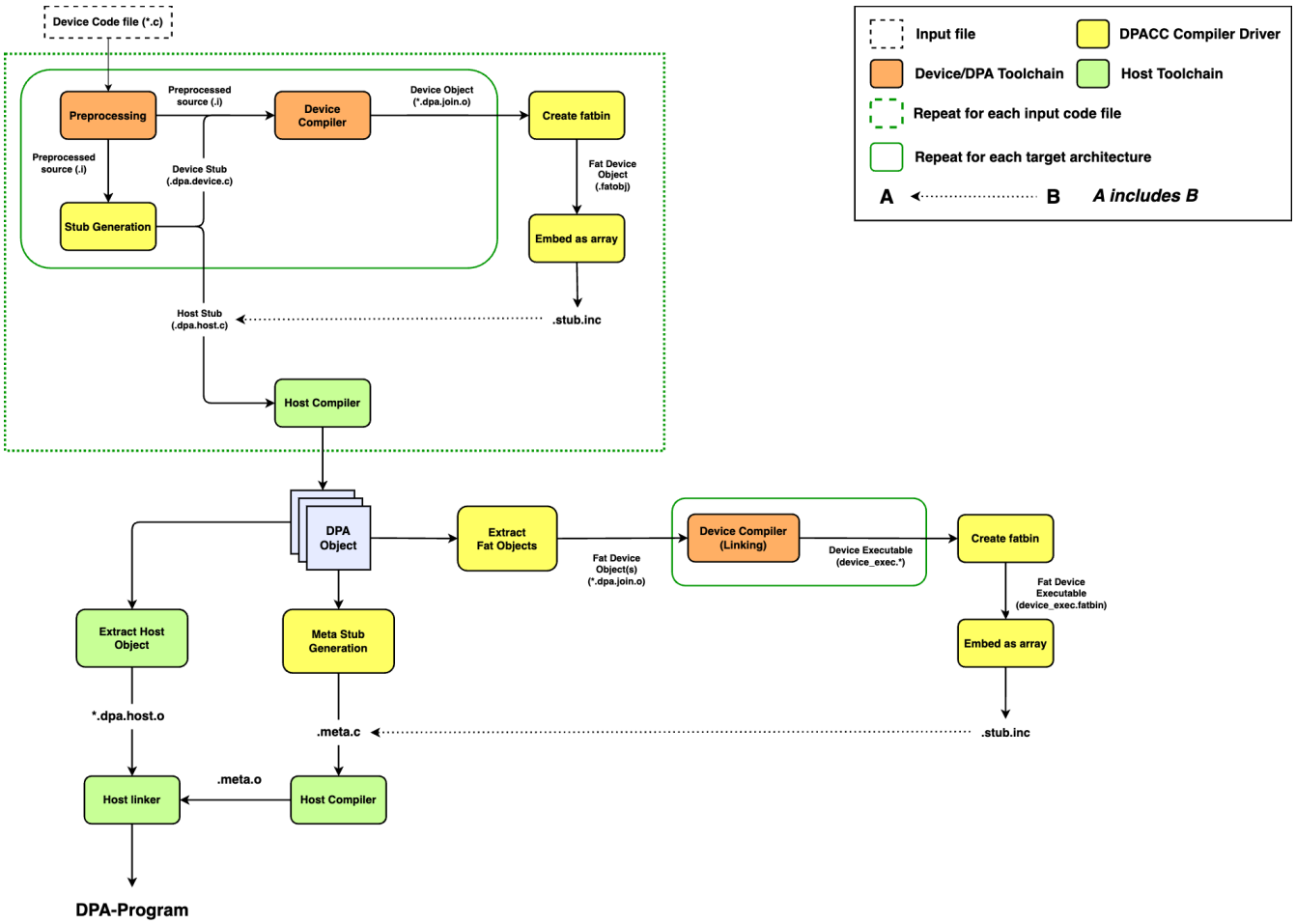
The host library is a static archive. The device library is a file of specialised fatbinary container format type which contains archives of device objects built for different targets.



The dark-grey box in above picture indicates the fatbinary container. The fatbinary container should be treated as an opaque object by the end-user and should not be manipulated in any way.

DPACC Trajectory

The following diagram illustrates DPACC compile-and-link mode trajectory.



Modes of Operation

In each of the below modes, DPACC accepts single or multiple target names through the 'mcpu' option and builds the output such that all the mentioned targets are supported.

Compile-and-link Mode

This is a one-step mode that accepts C source files or DPA object files and produces the DPA program. Specifying the output library name is mandatory in this mode.

Example commands:

```
$ dpacc in1.c in2.c -o myLib1.a -hostcc=gcc -mcpu=nv-dpa-bf3
                          # Takes C sources to produce myLib1.a
library which supports a single target - nv-dpa-bf3
```

```
$ dpacc in3.dpa.o in4.dpa.o -o myLib2.a -hostcc=gcc -mcpu=nv-dpa-bf3,nv-dpa-cx8            # Takes DPA object files to produce myLib2.a library which supports multiple targets - nv-dpa-bf3 and nv-dpa-cx8
$ dpacc in1.c in3.dpa.o -o myLib3.a -hostcc=gcc -mcpu=nv-dpa-bf3,nv-dpa-cx7,nv-dpa-cx8     # Takes C source and DPA object to produce myLib3.a library which supports multiple targets - nv-dpa-bf3, nv-dpa-cx7 and nv-dpa-cx8
```

Compile-only Mode

This mode accepts C source code and produces `.dpa.o` object files. These files can be given to DPACC to produce the DPA program. The mode is invoked by the `--compile` or `-c` option.

The user can explicitly provide the output object file name using the `--output-file` or `-o` option.

Example commands:

```
$ dpacc -c input1.c -hostcc=gcc -mcpu=nv-dpa-cx7
# Produces input1.dpa.o which supports a single target - nv-dpa-cx7
$ dpacc -c input2.c -o myObj.dpa.o -hostcc=gcc -mcpu=nv-dpa-cx8,nv-dpa-cx7           # Produces myObj.dpa.o which supports multiple targets - nv-dpa-cx7 and nv-dpa-cx8
$ dpacc -c input3.c input4.c -hostcc=gcc -mcpu=nv-dpa-bf3,nv-dpa-cx7,nv-dpa-cx8      # Produces input3.dpa.o and input4.dpa.o which support multiple targets - nv-dpa-bf3, nv-dpa-cx7 and nv-dpa-cx8
```

Library Generation Mode

This mode accepts C source files or DPA object files and produces the DPA program. Specifying the output DPA library name is mandatory in this mode.

Example commands:

```
$ dpacc in1.c in2.c -o libdummy1 -hostcc=gcc -mcpu=nv-dpa-cx8 -
gen-libs                                    # Takes C sources to
produce a DPA-Library (libdummy1_host.a and libdummy_device.a
archives) which supports a single target - nv-dpa-cx8
$ dpacc in3.dpa.o in4.dpa.o -o libdummy2 -hostcc=gcc -mcpu=nv-
dpa-cx8,nv-dpa-bf3 -gen-libs                 # Takes DPA object
files to produce a DPA-Library (libdummy2_host.a and
libdummy2_device.a archives) which supports multiple targets -
nv-dpa-bf3 and nv-dpa-cx8
$ dpacc in1.c in3.dpa.o -o outdir/libdummy3 -hostcc=gcc -mcpu=nv-
dpa-bf3,nv-dpa-cx7,nv-dpa-cx8 -gen-libs   # Takes C source and
DPA object to produce a DPA-Library (outdir/libdummy3_host.a and
outdir/libdummy3_device.a archives) which supports multiple
targets - nv-dpa-bf3, nv-dpa-cx7 and nv-dpa-cx8
```

Execution

To execute DOCA DPACC compiler:

```
Usage: dpacc <list-of-input-files> -hostcc=<path> -mcpu=<targets>
[other options]
Helper Flags:
  -h, --help                          Print help information
about DPACC
  -V, --version                       Print DPACC version
information
```

```
  -v, --verbose                          List the compilation
commands generated by this invocation while also executing every
command in verbose mode
  -dryrun, --dryrun                      Only list the compilation
commands generated by DPACC, without executing them
  -keep, --keep                          Keep all intermediate files
that are generated during internal compilation steps in the
current directory
  -keep-dir, --keep-dir                  Keep all intermediate files
that are generated during internal compilation steps in the given
directory
  -optf, --options-file <file>,...       Include command line
options from the specified file
```

Mandatory Arguments

Flag	DPACC Mode	Description
List of one or more input files	All	List of C source files or DPA object file names. Specifying at least one input file is mandatory. A file with an unknown extension is treated as a DPA object file.
`-mcpu`, `--mcpu <targets>`	All	Specify the list of target DPA hardwares for code generation. (See DPA Hardware Architectures for more details) Multiple target names can be specified through this option. Supported values: `nv-dpa-bf3, nv-dpa-cx7, nv-dpa-cx8`

Flag	DPACC Mode	Description
`-hostcc`, `--hostcc <path>`	All	Specify the host compiler. This is typically the native compiler present on the host system. **Note** The host compiler used to link the host application with the DPA program must be link-compatible with the `hostcc` compiler provided here.
`-o`, `--output-file <file>`	Compile-and-link/library generation	Specify name and location of the output file.

Commonly Used Arguments

Tip

Use `--help` option for a list of all supported options.

Flag	Description
`-app-name`, `--app-name <name>`	Specify DPA application name for the DPA program. This option is required if multiple DPA programs are part of a host application because each DPA application must have a unique name. Default name is `__dpa_a_out`.
`-flto`, `--flto`	Enable link-time optimization (LTO) for device code. Specify this option during compilation along with an optimization level in

Flag	Description
	`devicecc-options`.
`-devicecc-options`, `--devicecc-options <options>`,...	Specify the list of options to pass to the device compiler.
`-devicelink-options`, `--devicelink-options <options>`,...	Specify the list of options to pass during device linking stage.
`-device-libs`, `--device-libs '-L<path> -l<name>'`,...	Specify the list of device libraries including their names (in `-l`) and their paths (in `-L`). FlexIO libraries are linked by default.
`-I`, `--common-include-path <path>`,...	Specify include search paths common to host and device code compilation. FlexIO headers paths are included by DPACC by default.
`-o`, `--output-file <file>`	Specify name and location of the output file. • Compile-only mode – name of the output DPA object file. If not specified, `.dpa.o` is generated for each `.c` file. • Compiler-and-link mode – name of the output DPA program. This is a mandatory option in compiler-and-link mode. • Library generation mode – name of the output library. This is a mandatory option for this mode. Output files `<name>_device.a` and `<name>_host.a` are generated.

Flag	Description
`-hostcc-options`, `--hostcc-options <options>`,...	Specify the list of options to pass to the host compiler.
`-gen-libs`, `--gen-libs`	Generate a DPA library from input files
`-ldoca_dpa`, `--ldoca_dpa`	Link with DOCA-DPA libraries

Note

Using machine dependent options through `-devicecc-options` to influence compiler code generation is not supported. Examples of unsupported options through `-devicecc-options` are `-mcpu, -march, -mabi`, etc...

Note

The `devicecc-options` option allows passing any option to the device compiler. However, passing options that prevent compilation of the input file may lead to unexpected behavior (e.g., `-devicecc-options="-version"` makes the device compiler print the version and not process input files).

Note

Incompatible options that affect DPA global function argument sizes during DPACC invocation and host application compilation may lead to undefined behavior during execution (e.g., passing `-hostcc-options="-fshort-enums"` to DPACC and missing this option when building the host application).

DPA Hardware Architectures

The table below mentions the DPA architectures, the associated values supported in the compiler through -mcpu option and the macros defined by the compiler to identify these architectures.

Hardware name	Value	Macro
ConnectX-7	nv-dpa-cx7	__NV_DPA_CX7
Bluefield-3	nv-dpa-bf3	__NV_DPA_BF3
ConnectX-8	nv-dpa-cx8	__NV_DPA_CX8

Since ConnectX-7 and Bluefield-3 share the same DPA hardware, nv-dpa-cx7 is treated as an alias of nv-dpa-bf3 by the compiler.

Link Compatibility

Only relocatable objects which are link-compatible are allowed to be linked together. Incompatible objects are errored out during linking. The toolchain version of the linker should be same as the toolchain version of the compiler which produced the objects.

If two architectures 'A' and 'B' are link-compatible and 'B' is newer than 'A', objects built for target 'A' can be linked to build an app for target 'B'. However, the inverse i.e. linking objects built for target 'B' to build an app for target 'A' is not valid.

Bluefield-3/ConnectX-7 and ConnectX-8 are link-compatible i.e. objects built for Bluefield-3/ConnectX-7 can be linked together to build an application for ConnectX-8.

Object Target ↓ \ Executable Target →	nv-dpa-bf3 / nv-dpa-cx7	nv-dpa-cx8
nv-dpa-bf3 / nv-dpa-cx7	Compatible	Compatible
nv-dpa-cx8	Incompatible	Compatible

Architecture Macros

The compiler defines identifier macros for each version of DPA hardware as described in DPA Hardware Architectures section. Each identifier macro will have a unique integer value which is strictly greater than that of macros for older DPA CPU models. Known aliases such as Bluefield-3 DPA and ConnectX-7 DPA share the same integer value. The macro __NV_DPA is defined to the value of current compilation target. This can be used to write device code specific to a DPA hardware generation as shown below:

```
#if __NV_DPA == __NV_DPA_BF3
// Code for Bluefield-3 here
#elif __NV_DPA > __NV_DPA_BF3
// Code for devices after Bluefield-3 here
#endif
```

Note: The ordering established by value of hardware version identifier macros do not imply an ordering of features supported by hardware. It is the user responsibility to ensure that features used in the code specific for a DPA version are actually supported on the hardware.

LTO Usage Guidelines

Restrictions

- Only the default linker script is supported with LTO

- Using options `-fPIC` / `-fpic` / `-shared` / `-mcmodel=large` through `-devicecc-options` is not supported when LTO is enabled
- Fat bitcode objects containing both LLVM bitcode and ELF representation are not supported
- Thin LTO is not supported

Compatibility

During compilation, LLVM generates the object as bitcode IR (intermediate representation) when LTO is enabled instead of ELF representation. The bitcode IR generated by the DPA compiler is only guaranteed to be compatible within the same version. The toolchain version of the compiler which builds the objects involved in link-time optimization (enabled with `-flto`) and the toolchain version of the linker which performs LTO must be the same.

Deprecated Features

- The '-ldpa' option which links with DOCA-DPA libraries is deprecated and will be removed in future releases. The option '-ldoca_dpa' is to be used instead of '-ldpa'.

Examples

This section provides some common use cases of DPACC and showcases the `dpacc` command.

Building Libraries

This example shows how to build DPA libraries using DPACC. Libraries for DPA typically contain two archives, one for the host and one for the device.

```
dpacc input.c -hostcc=gcc -mcpu=nv-dpa-bf3 -o lib<name> -gen-libs
```

```
-hostcc-options="-fPIC"
```

This command generates the output files `lib<name>_host.a` and `lib<name>_device.a`.

The host stub archive can be linked with other host code to generate a shared/static host library.

- Generating a static host library:

```
ar x lib<name>_host.a                    # Extract objects to
generate *.o
ar cr lib<name>.a <*src.host.o> *.o      # Generate final
static archive with all objects
```

- Generating a shared host library:

```
gcc -shared -o lib<name>.so <*src.host.o> -Wl,-whole-archive
-l<name>_host -Wl,-no-whole-archive       # Link the
generated archive to build a shared library
```

Linking with DPA Device Library

The DPA device library generated by DPACC using `-gen-libs` as part of a DPA library can be consumed by DPACC using the `-device-libs` option.

```
dpacc input.c -hostcc=gcc -mcpu=nv-dpa-bf3 -o libInput.a -device-
libs="-L <path-to-library> -l<libName>"
```

Enabling Link-time Optimizations

Link-time optimizations can be enabled using `-flto` along with an optimization level specified for device compilation.

```
dpacc input1.c -hostcc=gcc -mcpu=nv-dpa-bf3 -c -flto -devicecc-
options="-O2"
dpacc input2.c -hostcc=gcc -mcpu=nv-dpa-bf3 -c -flto -devicecc-
options="-O2"
dpacc -mcpu=nv-dpa-bf3 input1.dpa.o input2.dpa.o -hostcc=gcc -o
libInput.a
```

Including Headers

This example includes headers for device compilation using `devicecc-options` and host compilation using `hostcc-options`. You may also specify headers for any compilation on both the host and device side using the `-I` option.

```
dpacc input.c -hostcc=gcc -mcpu=nv-dpa-bf3 -o libInput.a -I
<common-headers-path> -devicecc-options="-I <device-headers-
path>" -hostcc-options="-I <host-headers-path>"
```

Dump targets supported by a fatbinary file

The targets supported by a fatbinary file can be dumped using dpa-fatbin tool. This can be used to dump the targets supported by a DPA-Library from the device archive.

```
dpa-fatbin --list libfoo_device.a
dpa-fatbin --list device_exec.fatbin
```

Dump target of device ELF file

The target for which a device ELF file is built can be dumped using dpa-objdump tool.

```
dpa-objdump --file-headers foo.o
```

Generating output as source code

DPACC provides an option '-src-output' to generate the output as host source code. This source can be compiled by the host compiler to generate functionally equivalent output which DPACC would have generated directly.

This example shows how to build various outputs of DPACC as source using this option and how to compile the generated source.

DPA-Program source

Generate DPA-Program source passing this option to DPACC

```
dpacc input.c -hostcc=gcc -mcpu=nv-dpa-bf3 -o libfoo.c -src-
output
```

Compile the generated source using host compiler to generate an object and build an archive with this object. A macro __DPACC_SRC_TARGET__ needs to be defined when building this object to remove unnecessary code which is not needed when building from source.

```
$ gcc libfoo.c -c -I /opt/mellanox/flexio/include -Wno-attributes
-Wno-pedantic -Wno-unused-parameter -Wno-return-type -Wno-
implicit-function-declaration -D__DPACC_SRC_TARGET__
$ ar cr libfoo.a libfoo.o
```

DPA-Library source

Generate DPA-Library source passing this option to DPACC

```
dpacc input.c -hostcc=gcc -mcpu=nv-dpa-bf3 -o libfoo -gen-libs -
src-output
```

This generates the device archive libfoo_device.a and host code files libfoo.lib.c, input.dpa.c. The host archive of DPA-Library is generated by compiling these sources and building an archive. The __DPACC_SRC_TARGET__ macro needs to be defined in this case to remove unnecessary code.

```
$ gcc libfoo.lib.c input.dpa.c -c -I /opt/mellanox/flexio/include
-Wno-attributes -Wno-pedantic -Wno-unused-parameter -Wno-return-
type -Wno-implicit-function-declaration -D__DPACC_SRC_TARGET__
$ ar cr libfoo_host.a libfoo.lib.o input.dpa.o
```

DPA-Object source

Generate DPA-Object source passing this option to DPACC

```
dpacc input.c -hostcc=gcc -mcpu=nv-dpa-bf3 -c -src-output
```

This generates a single file input.dpa.c. Compile the host file to generate an object.

```
gcc input.dpa.c -c -I /opt/mellanox/flexio/include -Wno-
attributes -Wno-pedantic -Wno-unused-parameter -Wno-return-type -
Wno-implicit-function-declaration
```

DPA Compiler Usage

DPA-Compiler is an LLVM based compiler which is used by DPACC internally for compiling and linking the device code files. User specified options can be passed to the compiler and linker through the DPACC options `'--devicecc-options'` and `'--devicelink-options'` respectively.

Refer to the following resources for options which can be passed:

- *Clang Compiler User's Manual*
- *Clang command line argument reference*
- *Target-dependent compilation options*
- *LLD command line reference*

Note

Invoking the compiler, assembler, or linker directly may lead to unexpected errors.

Note

Linker options are provided through the compiler driver dpa-clang.

Note

The LLD linker script is honored in addition to the default configuration rather than replacing the whole configuration like in GNU ld. Hence, additional options may be required to override some default behaviors.

> **Note**
>
> Enabling optional extensions in standard library by defining _ _STDC_WANT_LIB_EXT1_ _ macro is not supported

dpacc-extract Command Line Options

`dpacc-extract` is a tool for extracting a device executable out of a DPA program or a host executable containing DPA program(s).

To execute dpacc-extract:

```
Usage: dpacc-extract <input-file> -o=<output-file> [other
options]
Helper Flags:

  -o, --output-file                  Specify name of the output
file
  -app-name, --app-name <name>       Specify name of the DPA
application to extract
  -mcpu, --mcpu <target>             Specify name of the device
for which the application is to be extracted
  -h, --help                         Print help information
about dpacc-extract
  -V, --version                      Print dpacc-extract version
information
  -optf, --options-file <file>,...   Include command line
options from the specified file
```

Mandatory arguments:

Flag	Description
Input file	DPA program or host executable containing DPA program. Specifying one input file is mandatory.
`-o`, `--output-file <file>`	Specify name and location of the output device executable.
`-app-name`, `--app-name <name>`	Specify name of the DPA application to extract. Mandatory if input file has multiple DPA apps.
`-mcpu`, `--mcpu <target>`	Specify name of the device for which the application is to be extracted. Mandatory if there are multiple target variants for an app.

Objdump Command Line Options

The dpa-objdump utility prints the contents of object files and final linked images named on the command line.

For more information, please refer to the Objdump command line reference.

Archiver Command Line Options

dpa-ar is a Unix ar-compatible archiver.

For more information, please refer to the Archiver command line reference.

NM Tool Command Line Options

The dpa-nm utility lists the names of symbols from object files and archives.

For more information, please refer to the NM tool command line reference.

Miscellaneous Notes

- Objects produced by LLD are not compatible with those generated by any other linker.
- Ensure that there is at least one reference to the device entry point function in the DPA-Program from the host application so that during linking, the DPA-Program is not silently discarded by the host linker because it is not referenced.

Release Notes

Changes in DPACC 1.10.0

New Features

- Support for fatbinaries where DPACC accepts multiple targets through 'mcpu' option
- Enforced linking policy where only compatible objects can be linked together
- dpa-objdump infers the target automatically without the need to explicitly specify 'mcpu' option
- Set FLEXIO_VER_USED in generated host stubs
- Support for new builtin __dpa_thread_l1_flush on nv-dpa-cx8 target

Limitations

- DPACC generates a warning about unknown target when building a DPA-Library from DPA-Objects produced by v1.9.0 or older

DOCA DPA Execution Unit Management Tool

This document describes the DPA Execution Unit (EU) management tool, `dpaeumgmt` .

> **Note**
>
> Execution unit partitions will be supported in future releases.

Introduction

This table introduces important terms for understanding this document:

Term	Definition
DPA	Data-path accelerator; an auxiliary processor designed to accelerate data-path operations.
DPA partition manager	PCIe device function capable of controlling the entire system's EUs. On NVIDIA® BlueField®-3 it is the ECPF. The DPA partition manager is by default associated with the default partition.
EU	Hardware execution unit; a logical DPA processing unit.
EU group	Collection/subset of EUs which could be created using `dpaeumgmt`. EU groups are created under an EU partition and could only be formed from the pool of EUs under that partition.
EU object	EU partition or EU group.
EU partition	An isolated pool of EUs which may be created using `dpaeumgmt`. Only when a partition is created and associated with other vHCAs are they able to use hardware resources and execute a DPA software thread.

Term	Definition
EU affinity	The method by which a DPA thread is paired with a DPA EU. DPA supports three types of affinity: • `none` – selects an EU from a pool of all available EUs • `strict` – select only the specified EU (by ID) • `group` – select an EU from all the EUs in the specified group

The DPA EU management tool can run either on the host machine or on the target DPU and allows users to manage the DPA's EUs which are the basic resource of the DPA. The tool enables the resource control of EUs to optimize computation resources usage of the DPA before using DOCA FlexIO SDK API.

Without EU allocation, a DPA software thread would lack access to the hardware pipeline/CPU time resource, and consequently not be able to execute.

`dpaeumgmt` serves the following main usages:

- Running a DPA software thread with `strict` affinity on a DPA EU (i.e., running a DPA thread using only the specific preselected EU). For this purpose, `dpaeumgmt` provides an option to query the maximum EU ID allowed to use.
- Allowing a DPA software thread to run over a DPA EU from a group of EUs:
  - Once an EU group is created, it is allocated a subset of EUs.
  - `dpaeumgmt` provides an ID to the created group which can be used to run DPA applications with `group` affinity where the affinity ID would be the same as that group's ID.
- EU partition management - the ability to manage EU partitions.

When the software stack wishes to run a DPA thread with `group` affinity type, one of the available EUs from the group's collection is used for the execution.

Note

A DPA thread may execute if and only if there is an available EU for it.

Execution Unit Objects

Upon boot, a default EU partition is automatically created. The default EU partition possesses all the system's EUs. The DPA partition manager function is the only function that belongs to it and can therefore control the entire resources of the system.

When running a DPA thread with `none` affinity, the EU chosen for the DPA thread to run with comes from the partition's pool of EUs. Namely, from the EUs belonging only to the DPA device's current partition which were not assigned to any EU groups (on the current partition). If the aforementioned group of EUs (i.e., the partition's default EU group) is empty, the DPA thread would fail to run with `none` affinity.

dpaeumgmt Commands

`dpaeumgmt` enables users to create, destroy, and query EU objects.

Note

`dpaeumgmt` tool must run with root privileges and users must execute `sudo mst start` before using it.

Top-level `dpaeumgmt` command syntax:

```
Usage: dpaeumgmt {help|version|eu_group|partition}

Type "./dpaeumgmt help" for detailed help
```

General Commands

- Print basic usage information for the tool:

```
dpaeumgmt -h
```

- Print a detailed help menu of the tool's commands:

```
dpaeumgmt help
```

- Print version information:

```
dpaeumgmt version
```

Execution Unit Group Commands

The commands listed in the following subsections are used to configure EU groups.

EU Group Command Flags and Arguments

The following table lists the flags relevant to `eu_group` commands. Arguments for the flags must be used within quotes (if more than one) and without extra spaces.

Short Option	Long Option	Description
`-h`	`--help`	Print out basic tool usage information.
`-d`	`--dpa_device`	The device interface name (MST/PCI/RDMA/NET).

Short Option	Long Option	Description
`-r`	`--range_eus`	The range of EUs to allocate an EU group or a partition. The argument must be provided within quotes.
`-g`	`--id_group`	Group ID number. This number must be positive and less than or equal to the `max_num_dpa_eu_group` parameter which may be retrieved using the command eu`_group info -d <device>`.
`-n`	`--name_group`	Group name; 15-character string. The argument must be provided within quotes.
`-f`	`--file_groups`	Full path or only the filename if it is located in the same directory as the executable directory (where `dpaeumgmt` is).

Info EU Group

Print information on the relevant DPA resources for the EU groups:

```
dpaeumgmt eu_group info --dpa_device <device>
```

Example:

```
$ sudo ./dpaeumgmt eu_group info -d mlx5_0
Max number of DPA EU groups: 15
Max number of DPA EUs in one DPA EU group: 190
Max DPA EU number available to use: 190
Max EU group name length is 15 chars
```

Create EU Group

Create an EU group with the specified name on the provided device's partition. The EUs indicated by the range are taken from the DPA device's EU partition.

```
dpaeumgmt eu_group create --dpa_device <device> --name_group
<name> --range_eus <range>
```

Example:

```
$ sudo ./dpaeumgmt eu_group create -d mlx5_0 -n "HG hello world1"
-r "6-8,16,55,70"
Group created successfully-
EU group ID: 1
EU group name: HG hello world
Member EUs are: 6-8,16,55,70
```

> **Note**
>
> After successfully creating an EU group, users can run a DPA thread using `group` affinity with the affinity type set to the group's ID.

Destroy EU Group

Destroy an EU group that exists on the device's partition with either the provided group name or ID.

```
dpaeumgmt eu_group destroy --dpa_device <device> [--name_group
```

```
<name> | --id_group <id>]
```

Example:

```
$ sudo ./dpaeumgmt eu_group destroy -d mlx5_0 -g 1
Group with group id: 1, was destroyed successfully
```

Query EU Group

Query EU groups residing on the provided device's partition. If one of the optional parameters is used, the command only queries the specific group and prints it if it exists:

```
dpaeumgmt eu_group query --dpa_device <device> [--name_group
<name> | --id_group <id>]
```

Example:

```
$ sudo ./dpaeumgmt eu_group query -d mlx5_0
1) EU group ID: 1
EU group name: HG hello world
Member EUs are: 6-8,16,55,70

In total there are 1 EU groups configured.
```

More options:

```
$ sudo ./dpaeumgmt eu_group query -d mlx5_0 -n "HG hello world"
$ sudo ./dpaeumgmt eu_group query -d mlx5_0 -g 1
```

Apply EU Group

Apply the EU groups provided in the file on the device's partition:

```
dpaeumgmt eu_group apply --dpa_device <device> --file_groups
<file>
```

File format example:

```
{
        "eu_groups": [
                { "name": "hg1", "range": "178-180"},
                { "name": "hg2", "range": "2-10"}
        ]
}
```

> **Note**
>
> The command removes all the previous EU groups defined on the EU partition that the DPA device belongs to and applies the ones from the file.

Example:

```
$ sudo ./dpaeumgmt eu_group apply -d mlx5_0 --file_groups
example.json
1) EU group ID: 1
EU group name: hg1
```

```
Member EUs are: 178-180

1) EU group ID: 2
EU group name: hg2
Member EUs are: 2-10

In total there are 2 EU groups configured.
```

EU Partition Commands

The commands listed in the following subsections are used to configure EU partitions.

EU Partition Command Flags and Arguments

The following table lists the flags relevant to EU `partition` commands. Arguments for the flags must be used within quotes (if more than one) and without extra spaces.

Short Option	Long Option	Description
`-h`	`--help`	Print out basic tool usage information.
`-d`	`--dpa_device`	The device interface name (MST/PCI/RDMA/NET).
`-r`	`--range_eus`	The range of EUs to allocate an EU group or a partition. The argument must be provided within quotes.
`-p`	`--id_partition`	Partition ID number. This number must be positive and less than or equal to the value of `max_num_dpa_eu_partition` which may be retrieved using the command `partition info -d <device>`.
`-v`	`--vhca_list`	The vHCA IDs to be associated with the partition. The argument must be provided within quotes.

Short Option	Long Option	Description
`-m`	`--max_num_eu_group`	The number of EU groups to reserve for the partition upon its creation.

Info EU Partition

Print the relevant DPA resources of the EU partitions:

```
dpaeumgmt partition info --dpa_device <device>
```

Example:

```
$ sudo ./dpaeumgmt partition info -d mlx5_0
Max number of DPA EU partitions: 15
Max number of VHCAs associated with a single partition: 32
Max number of DPA EU groups: 15
Note- an allocation of a partition consumes from the number of
DPA EU *groups* available to create
Max DPA EU number available to use: 190
```

Create EU Partition

Create an EU partition on the DPA device:

```
dpaeumgmt partition create --dpa_device <device> --vhca_list
<id_list> --range_eus <range> --max_num_eu_group <max_num>
```

Example:

```
$ sudo ./dpaeumgmt partition create -d mlx5_0 -v 1 -r 10-20 -m 2
Partition created successfully-
EU Partition ID: 1
Maximal number of groups: 2
The partition has a total of 1 associated VHCA IDs, namely: 1
Partition's member EUs are: 10-20
```

Destroy EU Partition

Destroy an EU partition that exists on the device's partition:

```
dpaeumgmt partition destroy --dpa_device <device> --id_partition
<id>
```

Example:

```
$ sudo ./dpaeumgmt partition destroy -d mlx5_0 -p 1
Partition with partition id: 1, was destroyed successfully
```

Query EU Partition

Query EU partitions that reside on the provided device's partition and print out the partition if it exists:

```
dpaeumgmt partition query --dpa_device <device> [--id_partition
<id>]
```

Example:

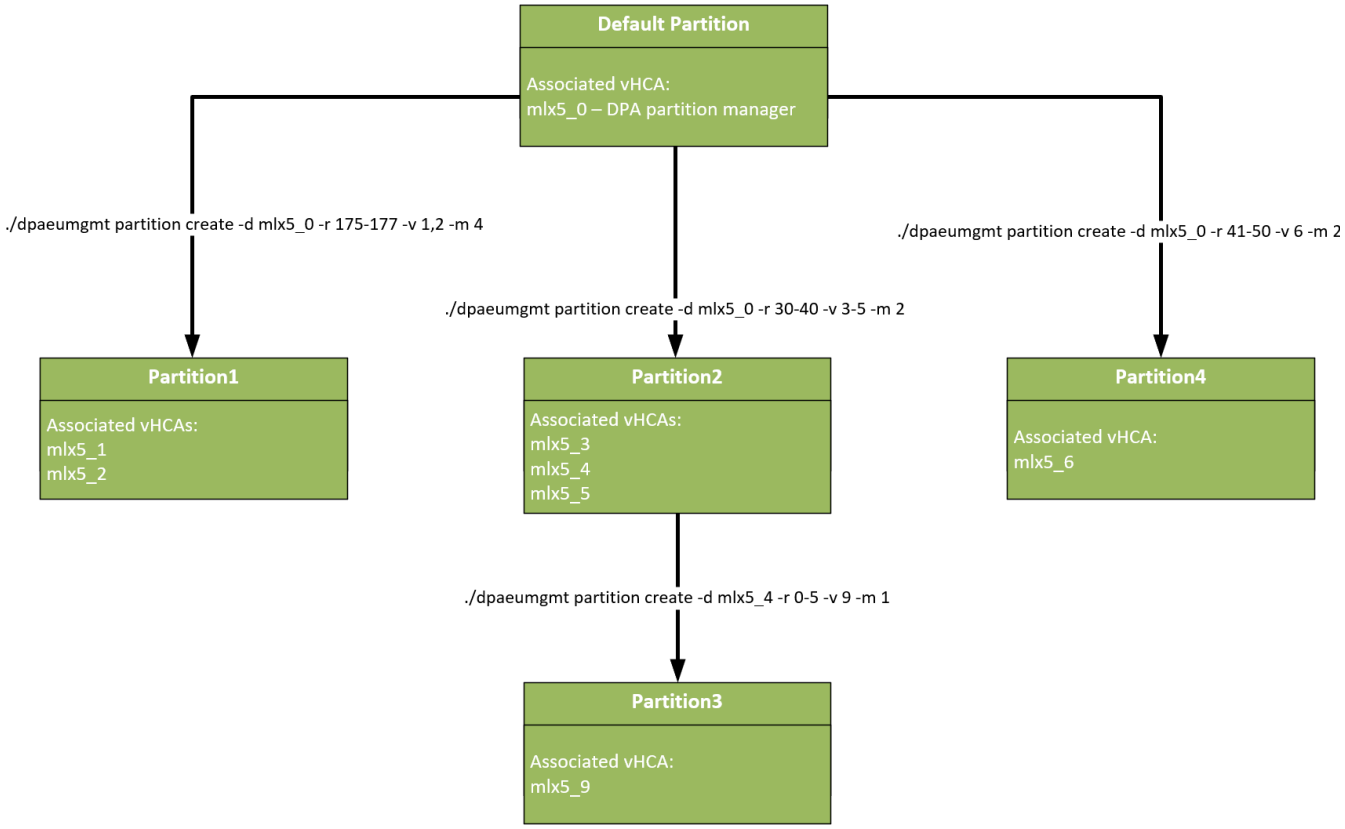
```
$ sudo ./dpaeumgmt partition query -d mlx5_0 -p 1
EU Partition ID: 1
Maximal number of groups: 2
The partition has a total of 1 associated VHCA IDs, namely: 1
Partition's member EUs are: 10-20
```

More options:

```
$ sudo ./dpaeumgmt partition query -d mlx5_0
```

vHCAs and Partitions

The following diagram illustrates the ownership and control of a partition by a vHCA and also which vHCAs have claim to (i.e., can use) a partition.



Known Limitations

- `dpaeumgmt` should run before creating a DPA process so all resources are configured ahead of time
    - Running the tool over a device with an existing DPA process results in failure
- The EU group name assigned by the user must be unique for every EU group on a specific partition or the EU group create command fails
- The creation of an EU partition consumes from the number of EU groups allowed on the vHCA's partition it is created on:
    - 1 group for the partition itself due to a default group created for each partition
    - `<max_num>` of groups which is the user's input provided upon partition creation
- Creating groups or running DPA threads in general (with any affinity) on interfaces other than ECPF, requires a configuration of a valid partition for the specific vHCA

- Only the default partition is exposed to the real EU numbers, all other partitions the user creates use virtual EUs
    - For example, if a user creates a partition with the range of EUs 20-40, querying the partition info from one of its virtual HCAs (vHCAs) would display EUs from 0-20. Therefore, the EU whose real number is 39 in this example would correspond to the virtual EU number 19.
- Group IDs on a non-default partition are virtual.
    - Different partitions can have completely distinct groups, even if they have the same ID.
    - The affinity ID parameter, specified on the FlexIO API, can distinguish between the groups according to the vHCA an application is running on.
- vHCA ID overlap is not allowed on EU partitions
- It is not possible to query vHCA IDs with `dpaeumgmt`, these are assumed to be known by the user beforehand
- Partition destruction fails if there are EU objects that exist on that partition
- It is not possible to know which EU has been chosen to run on
- Every vHCA sees the partition it belongs to, and its resources, as the entire world. It only sees:
    - Groups and partitions it created
    - The number of EUs it was given
    - The `max_num_eu_group` of the partition it belongs to
- No guarantee regarding EU group ID that will be given on group creation
- The default groups (of every partition) cannot be managed by the user
- The EU numbers available are between 0 and the max DPA EU number available to use minus 1 (the upper limit can be queried using the info command specified above)
- `dpaeumgmt` does not support virtual functions (VFs)

- It is not possible to create partitions on other vHCAs other than the DPA partition manager function
- There are at most 16 hardware EU group entities

DOCA DPA GDB Server Tool

This document describes the DPA GDB Server tool.

> **Info**
>
> The DPA GDB Server Tool is currently supported at beta level.

Introduction

The DPA GDB Server tool (`dpa-gdbserver`) enables debugging FlexIO DEV programs.

DEV programs for debugging are selected using a token (8-byte value) provided by the FlexIO process owner.

> **Info**
>
> Any GDB, familiar with RISC-V architecture, can be used for the debug.
> Refer to this page for information how to work with GDB.

Glossary

Term	Description
PUD	Process under debug. DEV-side processes intended for debug.
EU	Execution unit (similar to hardware CPU core)
DPA	Data path accelerator
RPC	Remote process communication. Mechanism used in FlexIO to run DEV-side

Term	Description
	code instantly. Runtime is limited to 6 seconds.
HOST	x86 or aarch64 Linux OS which manages dev-side code (i.e., DEV)
DEV	RISC-V code, loaded by HOST into the DPA's device. Triggered to run by different types of interrupts. DEV side is directly connected to ConnectX adapter card.
GDB	GNU Project debugger. Allows users to monitor another program while it executes.
GDBSERVER	Tool for remote debug programs
RTOS	Real-time operation system running on RISC-V core. Manages handling of interrupts and calls to DEV user processes routines.
RSP	Remote serial protocol. Used for interaction between GDB and GDBSERVER.

Known Limitations

- DPA GDB technology does not catch fatal errors. Therefore, if a fatal error occurs, core dump (created by `flexio_coredump_create()`) should be used.
- DPA GDB technology does not support Outbox access. GDB users cannot write to Doorbell or to Window configuration areas.
- DPA GDB technology does not support Window access. Read/write to Window memory does not work properly.

DPA-specific Notes

Token

The process under debug (PUD) can expose a debugging token. Every external process, using this token, get full access to the process with given token. To not show it constantly (e.g., for security reasons), users can modify their host application temporary. See `flexio_process_udbg_token_get()`.

Connection on Application Launch

If the code which needs debugging begins to run immediately after launch, the user should modify the host application to stop upon start to give the user time to run `dpa-gdbserver`. One possible way of doing this is to place function `getchar()` immediately after process creation.

Dummy Thread Concept

Something to consider with DPA debugging is that a PUD does not have a running thread all time (e.g., the process's thread may exist but be waiting for incoming packets). In a regular Linux application, this scenario is not possible and GDB does not support such cases.

Therefore, when no thread is running, `dpa-gdbserver` reports a dummy thread:

```
(gdb) info thread
  Id   Target Id                                    Frame
* 1    Thread 1.805378433 (Dummy Flexio thread) 0x0800000000000000 in ??
()
(gdb)
```

In this case user can inspect memory, create breakpoints, and give the `continue` command.

Commands like `step`, `next`, and `stepi` can not be executed for the Dummy thread.

Watchdog Issues

The RTOS has a watchdog timer that limits DEV code interrupt processes to 120 seconds. This timer is stopped when the user connects to DEV with GDB. Therefore users will have no time limitation for debugging.

Tool TCP Port and Execution Unit (EU)

By default, `dpa-gdbserver` uses TCP port 1981 and runs on EU 29. If this conflicts with another application (or if other instances of `dpa-gdbserver` are running), users should change the defaults as follows:

```
$> dpa-gdbserver mlx5_0 -T <token> -s <port> -E <eu_id>
```

Debugging

Preparation for Debug

Modify your FlexIO application if needed. Make sure the HOST code prints `udbg_token` and waits for GDB connection if needed:

```
+       uint64_t udbg_token;

        flexio_process_create(..., &flexio_process);

+       udbg_token =
flexio_process_udbg_token_get(flexio_process);
+       if (udbg_token)
+               printf("Process created. Use token >>> %#lx <<< for debug\n",
udbg_token);

+       printf("Stop point for waiting of GDB connection. Press Enter to continue..."); /* Usually
you don't need this stop point */
+       fflush(stdout);
+       getchar();
```

Extract the DPA application from the FlexIO application. For example:

```
$> dpacc-extract cc-host/app/host/flexio_app_name -o
flexio_app_name.rv5
```

Start Debugging

1. Run your FlexIO application. It should expose the debug token:

```
$> flexio_app_name mlx5_0
Process created. Use token >>> 0xd6278388ce4e682c <<< for
debug
```

2. Run `dpa-gdbserver` with the debug token received:

```
$> dpa-gdbserver mlx5_0 -T 0xd6278388ce4e682c
Registered on device mlx5_0
Listening for GDB connection on port 1981
```

3. Run any GDB with RISC-V support. For example, `gdb-multiarch`:

```
$> gdb-multiarch -q flexio_app_name.rv5
Reading symbols from flexio_app_name.rv5...
(gdb)
```

4. Connect to the gdbserver using proper TCP port and hostname, if needed:

```
(gdb) target remote :1981
Remote debugging using :1981
```

```
0x0800000000000000 in ?? ()
```

DPA-specific Debugging Techniques

Easy Example of Transitioning from Dummy to Real Thread

Transitioning between the dummy thread and a real thread is not standard practice for debugging under GDB. In an ideal situation, the user would know exactly the entry points for all their routines and can set breakpoints for all of them. Then the user may run the `continue` command:

```
(gdb) target remote :1981
Remote debugging using :1981
0x0800000000000000 in ?? ()
(gdb) info threads
  Id   Target Id                                    Frame
* 1    Thread 1.805378433 (Dummy Flexio thread) 0x0800000000000000 in ??
()
(gdb) b foo
Breakpoint 1 at 0x400000b2: file ../tests/path/hello.c, line 58.
(gdb) b bar
Breakpoint 2 at 0x40000518: file ../tests/path/hallo.c, line 113.
(gdb) continue
Continuing.
```

Initiate interrupts for your DEV program (depends your task), and GDB should catch a breakpoint and now the real thread of the PUD appear instead of the dummy:

```
(gdb) continue
Continuing.
(gdb) [New Thread 1.2]
[New Thread 1.130]
```

```
[New Thread 1.258]
[New Thread 1.386]
[Switching to Thread 1.2]

Thread 2 hit Breakpoint 1, foo(thread_arg=9008)
    at ../tests/path/hello.c:58
58              struct host_data *hdata = NULL;
(gdb) info threads
  Id   Target Id                                      Frame
* 2    Thread 1.2 (Process 0 thread 0x1 GVMI 0)     foo (arg=9008)
at ../tests/path/hello.c:58
  3    Thread 1.130 (Process 0 thread 0x81 GVMI 0)  foo (arg=9264) at
../tests/path/hello.c:58
  4    Thread 1.258 (Process 0 thread 0x101 GVMI 0) foo (arg=9648) at
../tests/path/hello.c:58
  5    Thread 1.386 (Process 0 thread 0x181 GVMI 0) foo (arg=9904) at
../tests/path/hello.c:58
(gdb)
```

From this point, you may examine memory and trace your code as usual.

Complicated Example of Transitioning from Dummy to Real Thread

In a more complicated situation, the interrupt happens after GDB connection. In this case, the real thread should start running but cannot because the PUD is in HALT state. The user can type the command `info threads`, see new thread instead of the old dummy, and then switch to the new thread manually:

```
(gdb) target remote :1981
Remote debugging using :1981
0x0800000000000000 in ?? ()
(gdb) info threads
  Id   Target Id                                  Frame
```

```
* 1    Thread 1.805378433 (Dummy Flexio thread) 0x0800000000000000 in ??
()
(gdb) info threads
[New Thread 1.32769]
  Id   Target Id                                          Frame
  2    Thread 1.32769 (Process 0 thread 0x8000 GVMI 0) bar (arg=0xc0,
len=0)
    at /path/lib/src/stub.c:167

The current thread <Thread ID 1> has terminated.  See `help
thread'.
(gdb) thread 2
[Switching to thread 2 (Thread 1.32769)]
#0  bar (arg=0xc0, len=0)
    at /path/lib/src/stub.c:167
167     {
(gdb) bt
#0  bar (arg=0xc0, len=0)
    at /path/lib/src/stub.c:167
#1  0x000000004000017a in foo (thread_arg=3221)
    at ../path/dev/hello.c:182
#2  0x0000000000000000 in ?? ()
Backtrace stopped: frame did not save the PC
(gdb)
```

Note

The same command `info threads` in lines 4 and 7 gives different results. This happens because the interrupt occurs between the instances and the real code begins to run.

The user must switch to the new thread manually (see line 14). After this, they can trace/debug the flow as usual (i.e., using the commands `step`, `next`, `stepi`).

Finishing Real Thread without Finishing PUD

Every interrupt handler at some point finishes its way and returns the CPU resources to RTOS. The most common way to do this is to call function `flexio_dev_thread_reschedule()`. The command `next` on this function will have the same effect as the command `continue`:

```
205             __dpa_thread_fence(__DPA_MEMORY, __DPA_W,
__DPA_W);
(gdb) next
206             flexio_dev_cq_arm(dtctx,
app_ctx.rq_cq_ctx.cq_idx, app_ctx.rq_cq_ctx.cq_number);
(gdb) next
208             if ((dev_errno =
flexio_dev_get_and_rst_errno(dtctx))) {
(gdb) next
213             print_sim_str("Nothing to do. Wait for next duar\n", 0);
(gdb) next
214             flexio_dev_thread_reschedule();
(gdb) next
```

> **Info**
>
> GDB waits until the user types `^C` or a breakpoint is reached after the next interrupt occurred.

Error Reporting

> **Info**
>
> The DPA GDB server tool has been validated with `gdb-multiarch` (version 9.2) and with GDB version 12.1 from RISC-V tool chain.

> **Note**
>
> The GDB server should support all commands described in GDB RSP (remote serial protocol) for GDB stubs. But only the most common GDB commands are supported.

Should a dpa-gdbserver bug occur, please provide the following data:

- Used GDB (name and version)
- Commands sequence to reproduce the issue
- DPA GDB server tool console output
- DPA GDB server tool log directory content (see next part for details)
- Optional – output data printed when `dpa-gdbserver` is run in verbose mode

Tool Log Directory

For every run, a temporary directory is created with the template `/tmp/flexio_gdbs.XXXXXX`.

To locate the latest one, run the following command:

```
$> ls -ldtr /tmp/flexio_gdbs.* | tail
```

Verbosity Level of gdbserver

By default, `dpa-gdbserver` does not print any log information to screen. Adding - `v` option to command line increases verbosity level, printing additional info to `dpa-gdbserver` terminal display. Verbosity level is incremented according to number of 'v' in command line switch (i.e. `-vv`, `-vvv` etc.).

One `-v` shows the RSP exchange. This is a textual protocol, so users can read and understand requests from GDB and answers from the GDB server:

```
<<<<< "qTStatus"
>>>>> ""
<<<<< "?"
>>>>> "S05"
<<<<< "qfThreadInfo"
>>>>> "mp01.30011981"
<<<<< "qsThreadInfo"
>>>>> "l"
<<<<< "qAttached:1"
>>>>> "1"
<<<<< "Hc-1"
>>>>> "OK"
<<<<< "qC"
>>>>> "QCp01.30011981"
```

Info

In the examples, `<<<<<` and `>>>>>` are used to indicate data received from GDB and transmitted to GDB, respectively.

When running with a higher verbosity level (e.g., run `dpa-gdbserver` with option `-vv` or higher), the exchange with the RTOS module is shown:

```
<<<<< "qfThreadInfo"
/  2/dgdbs_handler - cmd 0x5
/  2/dgdbs_handler - retval 0x4
>>>>> "mp01.30011981"
<<<<< "qsThreadInfo"
/  2/dgdbs_handler - cmd 0x5
/  2/dgdbs_handler - retval 0x5
>>>>> "l"
<<<<< "m8000000000000000,4"
/  2/dgdbs_handler - cmd 0xc
/  2/dgdbs_handler - retval 0x9
>>>>> "E0a"
<<<<< "m7ffffffffffffc,4"
/  2/dgdbs_handler - cmd 0xc
/  2/dgdbs_handler - retval 0x9
>>>>> "E0a"
<<<<< "qSymbol::"
>>>>> "OK"
```

Info

Lines beginning with `/ #/` provide the number of internal RTOS threads printed from the DEV side.

Useful Info Regarding Work with GDB

This section provides useful information about commands and methods which can help users when performing DPA debug. This is not related to the `dpa-gdbserver` itself. But this is about remote debugging and FlexIO sources.

Command "directory"

GDB can run on a different host from the one where compilation was done. For example, users may have compiled and run their application on `host1` and run their instance of GDB on `host2`. In this case, users will see the error message `../xxx/yyy/zzz/your_file.c: No such file or directory`. To solve this problem, copy sources to the host running GDB (`host2` in the example). Make sure to save the original code hierarchy. Use GDB command `directory` to inform where the sources are to GDB:

```
host2~$> gdb-multiarch -q /tmp/my_riscv.elf
Reading symbols from /tmp/my_riscv.elf...
(gdb) b foo
Breakpoint 1 at 0x4000016c: file ../xxx/yyy/zzz/my_file.c, line 182.
(gdb) target remote host1:1981
Remote debugging using host1:1981
0x0800000000000000 in ?? ()
(gdb) c
Continuing.
[New Thread 1.32769]
[Switching to Thread 1.32769]

Thread 2 hit Breakpoint 1, foo (thread_arg=5728) at
../xxx/yyy/zzz/my_file.c:182
182     ../xxx/yyy/zzz/my_file.c: No such file or directory.
(gdb) directory /tmp/apps/
Source directories searched: /tmp/apps:$cdir:$cwd
(gdb) list
```

```
179             struct flexio_dev_thread_ctx *dtctx;
180             uint64_t dev_errno;
181
182             print_sim_str("=====> NET event handler started\n", 0);
183
184             flexio_dev_print("Hello GDB user\n");
185
```

Note

Pay attention to the exact path reported by GDB. The argument for the command `directory` should point to the start point for this path. For example, if GDB looks for `../xxx/yyy/zzz` and you placed the sources in local directory `/tmp/copy_of_worktree`, then the command should be `(gdb) directory /tmp/copy_of_worktree/xxx/` and not `(gdb) directory /tmp/copy_of_worktree/`.

Sometimes, the `*.elf` file provides a global path from the root. In this case, use the command `set substitute-path <from> <to>`. For example, if the file `/foo/bar/baz.c` was moved to `/mnt/cross/baz.c`, then the command `(gdb) set substitute-path /foo/bar /mnt/cross` instructs *GDB* to replace `/foo/bar` with `/mnt/cross`, which allows GDB to find the file `baz.c` even though it was moved.

See this page of GDB documentation for more examples of specifying source directories.

Core Dump Usage

If the code runs into a fatal error even though the host side of your project is implemented correctly, a core dump is saved which allows analyzing the core. It should

point exactly to where the fatal error occurred. The command `backtrace` can be used to examine the memory and its registers. Change the frame to see local variables of every function on the backtrace list:

```
$> gdb-multiarch -q -c crash_demo.558184.core /tmp/my_riscv.elf
Reading symbols from /tmp/my_riscv.elf...

[New LWP 1]
#0  0x000000004000126e in read_test (line=153, ptr=0x30) at
/xxx/yyy/zzz/my_file.c:109
109             val = *(volatile uint64_t *)ptr;
(gdb) bt
#0  0x000000004000126e in read_test (line=153, ptr=0x30) at
/xxx/yyy/zzz/my_file.c:109
#1  0x000000004000031a in tlb_miss_test (op_code=1) at
/xxx/yyy/zzz/my_file.c:153
#2  0x0000000040000144 in test_thread_err_events_entry_point
(h2d_daddr=3221258560) at /xxx/yyy/zzz/my_file.c:588
#3  0x00000000400013fc in
_dpacc_flexio_dev_arg_unpack_test_err_events_dev_test_thread_err_ev
(argbuf=0xc0008228, func=0x400000b0
<test_thread_err_events_entry_point>)
    at /tmp/dpacc_xExkvE/test_err_events_dev.dpa.device.c:67
#4  0x0000000040001680 in flexio_hw_rpc (host_arg=3221258752) at
/local_home/www/flexio-sdk/libflexio-
dev/src/flexio_dev_entry_point.c:75
#5  0x0000000000000000 in ?? ()
Backtrace stopped: frame did not save the PC
(gdb) frame 4
#4  0x0000000040001680 in flexio_hw_rpc (host_arg=3221258752) at
/local_home/igorle/flexio-sdk/libflexio-
dev/src/flexio_dev_entry_point.c:75
75                      retval = unpack_cb(&data_from_host-
>func_params.arg_buf,
(gdb) p /x *data_from_host
```

```
$2 = {poll_lkey = 0x1ff2b1, window_id = 0x3, poll_haddr =
0x55dc0f40b900, entry_point = 0x400013d8, func_params = {func_wo_pack =
0x0, dev_func_entry = 0x400000b0, arg_buf = 0xc0008140}}
(gdb)
```

Debug of Optimized Code

Usually highly optimized code is compiled and run.

Two types of mistakes in code can be considered:

- Logical errors
- Optimization-related errors

Logical errors (e.g., using `&` instead of `&&`) are reproduced on the non-optimized version of the code. Optimization related errors (e.g., forgetting volatile classification, non-usage of memory barriers) only impact optimization. Non-optimized code is much easier for tracing with GDB, because every C instruction is translated directly to assembly code.

It is good practice to check if an issue can be reproduced on non-optimized code. That helps observing the application flow:

```
$> build.sh -O 0
```

For tracing this code, using GDB commands `next` and `step` should be sufficient.

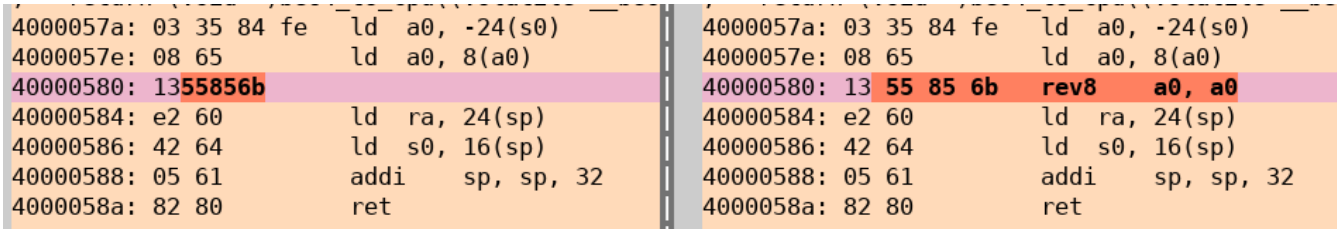
But if an issue can only be reproduced on on optimized code, you should start debugging it. This would require reading disassembly code and using the GDB command `stepi` because it becomes a challenge to understand exactly which C-code line executed.

Disassembly of Advanced RISC-V Commands

DPA core runs on a RISC-V CPU with an extended instruction set. The GDB may not be familiar with some of those instructions. Therefore, `asm` view mode shows numbers instead of disassembly. In this case it is recommended to disassemble your RISC-V binary code manually. Use the `dpa-objdump` utility with the additional option `--mcpu=nv-dpa-bf3`.

```
$> dpa-objdump -sSdxl --mcpu=nv-dpa-bf3 my_riscv.elf >
my_riscv.asm
```

The following screenshot shows the difference:



DOCA DPA PS Tool

Introduction

DOCA `dpa-ps` is a CLI tool which allows users to monitor running DPA processes and threads. The tool presents sorted lists of the currently running DPA processes and threads.

> **Info**
>
> The process ID output of the `dpa-ps` tool may be used as the input parameter for the `dpa-statistics` tool.

> **Info**
>
> This tool is supported for NVIDIA® BlueField®-3 only.

Command Flags and Arguments

The following table lists the flags for the `dpa-ps` tool .

Short Option	Long Option	Description
`-h`	`--help`	Help information
`-d`	`--device`	Device interface name (MST/RDMA)
`-p`	`--process-id`	Hexadecimal process ID for filtering
`-t`	`--threads`	Show threads info for each process

Short Option	Long Option	Description
`-i`	`--suppress-header-info`	Suppress print header info

> **Info**
>
> Arguments for the flags must be used within quotes (if more than one) and without extra spaces.

Example

```
$ sudo ./dpa-ps -d mlx5_0 -t
ProcessID    Process Name
    ThreadID     Thread Name
0            PROCESS_0
    5            EH_0_5_5
    6            EH_0_5_5
1            PROCESS_1
    3            EH_1_3_3
    4            EH_1_4_4
2            PROCESS_2
3            PROCESS_3
    0            EH_3_0_0
    1            EH_3_1_1
    2            EH_3_2_2
4            PROCESS_4
```

Known Limitations

- The `dpa-ps` and `dpa-statistics` tools cannot be run at the same time on the same device

DOCA DPA Statistics Tool

Introduction

DOCA `dpa-statistics` is a CLI tool which allows users to monitor and obtain statistics on thread execution per running DPA process and thread. The tool is used to expose information about the running DPA processes and threads and to collect statistics on DPA thread performance.

The tool presents performance information for running DPA threads, including the number of cycles and instructions executed in a time period. The tool enables initiating and stopping collection of statistics and displaying the data collected per thread.

Info

The process ID output of the `dpa-ps` tool may be used as the input parameter for the `dpa-statistics` tool.

Info

This tool is supported for NVIDIA® BlueField®-3 only.

Collecting Performance Statistics Data

The command `collect` works on four mutually exclusive modes:

- Enable mode – start collecting performance data
- Disable mode – stop collecting performance data

- Timeout mode – start collecting, wait with a timeout, stop collect and print info. User could break the wait with Ctrl-C command and then the timeout will be canceled and tool will disable statistics collection and prints the info with the actual time of the collect operation.

- Infinite mode – no special flags. Same as timeout mode but with infinite timeout. The tool awaits the Ctrl-C command to stop.

The following table lists the `collect` command's flags and arguments:

Short Option	Long Option	Description
`-h`	`--help`	Help information
`-d`	`--device`	Device interface name (MST/RDMA)
`-p`	`--process-id`	Hexadecimal process ID for filtering **Info** This flag indicates a specific command for the command to operate on. Otherwise, statistics are collected from all processes.
`-i`	`--suppress-header-info`	Suppress print header info
`-n`	`--enable`	Enable collect info
`-o`	`--disable`	Disable collect info
`-t`	`--timeout`	Enable collect, wait with timeout, disable collect and print info **Info**

Short Option	Long Option	Description
		Timeout value is in milliseconds. Examples for inputting timeout value: • 45 – 45 milliseconds • 45.55 – 45 milliseconds and 550,000 nanoseconds • .0005 – 500 nanoseconds • 45m55n – 45 milliseconds and 55 nanoseconds • 66n – 66 nanoseconds
`-r`	`--reset`	Reset counters before operation starting collect operation

Presenting Statistics List

Presenting performance statistics is applicable after initiating data collection.

The following table lists the `show` command's flags and arguments:

Short Option	Long Option	Description
`-h`	`--help`	Help information
`-d`	`--device`	Device interface name (MST/RDMA)
`-p`	`--process-id`	Hexadecimal process ID for filtering
`-i`	`--suppress-header-info`	Suppress print header info

Output example:

```
$ sudo ./dpa-statistics show -d mlx5_0 -p 1
ProcessID    Process Name
    ThreadID    Cycles             Instruction        Time
Executions         Thread Name
1            PROCESS_1
```

```
    3             266268              18193               164
41                  EH_1_0_0
    4             411571              32727               252
47                  EH_1_1_1
```

Where:

- `ProcessID` – The `dpa_process_object_id` to which the thread belongs
- `Process Name` – The `dpa_process_name` to which the thread belongs
- `ThreadID` – DPA thread object ID
- `Cycles` – Total EU cycles the thread used
- `Instruction` – Total number of instructions the thread executed
- `Time` – Total time in ticks the thread was active
- `Executions` – Total number of thread invocations
- `Thread Name` – The `dpa_thread_name`

Examples

- Example of `collect` in infinite mode for process 0 with suppress header info:

```
$ sudo ./dpa-statistics collect -d mlx5_0 -p 0 -i
...^C
Data collected for 4606 milliseconds 0 nanoseconds
0           PROCESS_0
    5           223964              13754               140
31          EH_0_5_5
    6           190130              13754               114
31          EH_0_6_6
```

- Example of `collect` in timeout mode with a timeout of 1 second and half a millisecond.

```
$ sudo ./dpa-statistics collect -d mlx5_0 -t 1000.500
Data collected for 1000 milliseconds 500000 nanoseconds
ProcessID    Process Name
    ThreadID    Cycles              Instruction         Time
Executions          Thread Name
0            PROCESS_0
    5           223964              13754               140
31                 EH_0_5_5
    6           190130              13754               114
31                 EH_0_6_6
1            PROCESS_1
    3           266268              18193               164
41                 EH_1_3_3
    4           411571              32727               252
47                 EH_1_4_4
2            PROCESS_2
3            PROCESS_3
    0           223205              13754               137
31                 EH_3_0_0
    1           189896              13754               113
31                 EH_3_1_1
    2           191796              13754               117
31                 EH_3_2_2
4            PROCESS_4
```

- Example of enabling statistics collection with reset of counters.

```
$ sudo ./dpa-statistics collect -d mlx5_0 -n -r
```

- Example of disabling statistics collection.

```
$ sudo ./dpa-statistics collect -d mlx5_0 -o
```

Known Limitations

- Reading large statistics counter blocks takes a long time
- The `dpa-ps` and `dpa-statistics` tools cannot be run at the same time on the same device

ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.
