SDx Command and Utility Reference Guide

UG1279 (v2018.3) January 24, 2019

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
01/24/2018 Version 2018.3	
Throughout the document	General updates
12/05/2018 Version 2018.3	
Chapter 2: SDx (sdx) Command Options	Updated options.
Chapter 2: SDx (sdx) Command Options	Updated options.
XOCC Common Options	Updated options.
XOCC Options for Compile Mode	Updated options.
XOCC Options for Link Mode	Updated options.
XP Parameters	Updated parameters.
System Options	Removed merge options, and created a new topic called Merge Options.
Merge Options	New topic.
Chapter 6: kernelinfo Utility	New topic.
Chapter 7: platforminfo Utility	New topic.
Chapter 8: xbutil (Xilinx Board Utility)	Updated list of options.
Chapter 9: package_xo Command	New topic.
Chapter 10: sdx_pack Utility	Updated options.
Chapter 11: xclbinutil Utility	New topic.
10/02/2018 Version 2018.2.xdf	
Throughout the document	Replaced xbsak with xbutil and removed xbinst.
xbinst	Removed chapter.
Chapter 8: xbutil (Xilinx Board Utility)	Updated chapter by removing Requirements for Boot Function of xbsak section and changed commands into sections.
XOCC Command Line Utility	Updated XP Parameters options table.
07/02/2018 Version 2018.2	
Throughout the document	General updates
XOCC Options for Compile Mode	Added information for these command options: • `--max_memory_ports` • `--memory_port_data_width`
XOCC Options for Link Mode	Added information for `--sys_config` command option.

Section	Revision Summary
General Options (to `sds++`)	Added information for these command options: • `-debug-xrf` • `-debug-xrf-cc`
Hardware Function Options (to `sds++`)	Added information for these command options: • `-hls-target` • `-hls-target-flags`
System Options (to `sds++`)	Added information for `-debug-port` command option.
Chapter 10: sdx_pack Utility	Added information for `-lib` command option.
06/06/2018 Version 2018.2	
Initial Xilinx release.	N/A

Table of Contents

Introduction

The Xilinx® SDx™ environment provides tools to build your design similar to a software-based design flow, where the source code is first compiled and then linked against each other. In addition, it provides numerous utilities to analyze generated files including a utility for board install and administration. These include:

- **XOCC Compiler** (`xocc`): The Xilinx OpenCL™ compiler (`xocc`) is a command line utility for compiling kernel accelerator functions and linking them with the SDAccel™ environment supported platforms.
- **SDSCC/SDS++ System Compilers** (`sdscc/sds++`): The SDSCC/SDS++ system compilers (`sdscc/sds++`, referred to as sds++) compile and link C/C++ source files into an application-specific hardware/software system-on-chip (SoC), targeting embedded Arm® Cortex™-A9, A53, and R5 CPUs with programmable logic hardware accelerators.
- **Emulation Configuration Utility** (`emconfigutil`): The emulation configuration utility (`emconfigutil`) is used to automate the creation of the emulation configuration file.
- **Xilinx Board Utility** (`xbutil`): The Xilinx Board Utility (`xbutil`) is a command line tool used to perform various board installation, administration, and debug tasks independent of the SDAccel runtime library, and for the SDAccel tools installation.

This document provides a reference for commands, syntax, and the various options that are available for each of the utilities. Some of these command settings can be configured through the SDx GUI as described in either *SDAccel Environment User Guide* (UG1023) or *SDSoC Environment User Guide* (UG1027).

SDx (sdx) Command Options

The `sdx` command is the primary entry into the SDx™ development environments. It provides options for specifying the workspace, and options of the project. The following are the options of the `sdx` command:

Display Options

The display options display the specified information intended for review:

- **`-help`:** Help -- display this message and quit.
- **`-version`:** Display Version and quit.
- **`-wait`:** Wait for SDx to complete.

Command Options

The command options specify how the `sdx` command is configured for the current workspace and project:

- **`-workspace <Workspace location>`:** Specify the Workspace directory for SDx projects
- **`{-lp <repository_path>}`:** Add `<repository_path>` to the list of Driver/OS/Library search directories.
- **`-report <report file>`:** Specify the report file to load in SDx GUI
- **`-builddir <build directory>`:**

  Specify the directory containing build results to import as an SDx Build Results project.

  This is typically the directory where you run the MAKE file.

- **`-rundir <run directory>`:**

  (Requires `-builddir`) Specify the directory containing reports generated by the host executable. Default: build directory.

  These reports will be included in the SDx Build Results project.

- **`-projectname <project name>`:** (Requires `-builddir`) Specify a name of the imported SDx Build Results project. Default: results_1

- `-eclipseargs <eclipse arguments>`:

  Eclipse-specifc arguments are passed to Eclipse.

- `-vmargs <java vm arguments>`:

  Additional arguments to be passed to Java VM.

XOCC (Xilinx OpenCL Compiler) Command Line Utility

The Xilinx® OpenCL™ Compiler (`xocc`) is a standalone command line utility for both compiling kernel accelerator functions and linking them with the SDAccel™ environment supported platforms. This section describes the `xocc` link and compile commands. All commands are provided in the following sections:

- XOCC Common Options

  Options common to both Compile and Link modes.
- XOCC Options for Compile Mode
- XOCC Options for Link Mode

The first step in building any system is to select an acceleration platform supported by Xilinx or third-party providers and to compile a kernel accelerator function using the `-c/--compile` option. The default output name for the `.xo` file is `a.xo`; rename the file to reflect the kernel name.

The `-c/--compile` command syntax is as follows:

```
xocc -c --platform <platform_name> <kernel_source_file> -o
<xo_kernel_name>.xo
```

TIP: *OpenCL uses the `kernel` keyword within the OpenCL file to identify a kernel. For C/C++ kernels, you need to provide the kernel name by `--kernel <kernel_name>`.*

The second step is to link one or more kernels into the platform to create the binary container `xclbin` file using the `-l/--link` option. The default output name for the `xclbin` file is `a.xclbin`; rename it as needed

The `-l/--link` command syntax is as follows:

```
xocc -l --platform <platform_name> <xo_kernel1_name>.xo \
[<xo_kernel2_name>.xo ..] -o <xclbin_name>.xclbin
```

For a list of supported platforms, see the release notes for the product you are using:

- For SDAccel, see the *SDAccel Environment Release Notes, Installation, and Licensing Guide* (UG1238).

- For SDSoC™, see the *SDSoC Environments Release Notes, Installation, and Licensing Guide* (UG1294).

IMPORTANT! *All provided examples included in the SDAccel installation use the Makefile to compile OpenCL applications with `xcpp` and `xocc` commands that can be used as references for compiling user applications.*

XOCC Common Options

IMPORTANT! *Do not mix targets between compilation and linking. For instance do not compile with `sw_emu` and link with `hw_emu`.*

Table 1: **XOCC Common Options (For Compile and Link Modes)**

Option	Valid Values	Description
`--platform <arg>`	Name of supported acceleration platform by Xilinx or full path to .xpfm file that represents a platform.	Required. The `--platform` option accepts either a platform name or alternatively an `xpfm` file name (using full or relative path) that represents the top level of a platform. This is needed when you use a platform that is not included by default in the SDAccel tool installation. Set the target Xilinx device. For example: `--platform xilinx_vcu1525_dynamic_5_1` For a list of all supported platforms and devices, see the SDAccel Product Page. When using a platform that is not included by default in the SDAccel tool installation, the `.xpfm` file that represents a platform should be provided using the full path.
`--target <arg>`	[`sw_emu` | `hw_emu` | `hw`]	Specifies a compile target. The compile target is specified with the `--target <sw_emu...>` option. The default compile target is `hw_emu`. • `sw_emu`: Software emulation • `hw_emu`: Hardware emulation • `hw`: Hardware Default: `hw`
`--output <arg>`	File name with `.xo` for compiling and `.xclbin` for linking, depending on `xocc` mode.	Optional. Sets output file name. Default: • `a.xo` for compile mode • `a.xclbin` for link and build mode
`--version`	N/A	Prints the version and build information of XOCC.
`--help`	N/A	Prints help.

Table 1: **XOCC Common Options (For Compile and Link Modes)** *(cont'd)*

Option	Valid Values	Description
`--kernel_frequency <arg>`	Frequency (MHz) of the kernel for single clock support. For multi-clock support, each clock ID and corresponding frequency must be entered. For example, xocc --kernel_frequency 0:300|1:500.	Sets a user-defined clock frequency (in MHz) for the kernel, overriding a default value from the hardware platform. RTL kernels support multi-clock: Syntax Example (overrides 1 clock) • xocc --kernel_frequency 300 • xocc --kernel_frequency 0:300 • DSA with one kernel clock: 300 MHz for KERNEL_CLK • DSA with two clocks: 300 MHz for DATA_CLK Syntax Example 2 (overrides more than 1 kernel clock) • xocc --kernel_frequency 0:300|1:500 DSA with two clocks (+ RTL kernel with two clock ports): • 300 MHz for DATA_CLK and 500 MHz for KERNEL_CLK2
`--profile_kernel <arg>`	`data:[ kernel_name | all ]:[ compute_unit_name | all ]:[ interface_name | all ](:[ counters | all ])` `[ stall | exec ]:[ kernel_name | all ]:[ compute_unit_name | all ](:[ counters | all ])`	Profiling DDR memory traffic for kernel and host. The last field for trace value (`counters` or `all`) is optional. If not specified, the default value is `all`. For `[ stall | exec ]`, the interface_name field is *not* supported. The `stall` option must be specified during `xocc` compile (`-c`) to direct HLS to enable stall signals before using this option during `xocc` link (`-l`).
`--xp <arg>`	Refer to XP Parameters.	Specifies detailed parameter and property settings in the Vivado® Design Suite used to implement the FPGA hardware. For example: `--xp <kernel_name>:stream` Familiarity with the Vivado Design Suite is recommended to make the most use of these parameters. For a complete description of the `--xp` option, see XP Parameters.
`--debug`	N/A	Generates code for debugging.
`--message-rules <arg>`	Message rule file name	Optional. Specifies a message rule file with message controlling rules. For more details, see Using the Message Rule File.
`--save-temps`	N/A	Saves intermediate files/directories created during the compilation and build process.
`--report_dir <arg>`	Directory	Specifies a report directory. If the `--report` option is specified, the default is to generate all reports in the current working directory (cwd). If no report directory is specified, the tool saves the files to `<cwd>/_x/reports`.

Table 1: **XOCC Common Options (For Compile and Link Modes)** *(cont'd)*

Option	Valid Values	Description
`--log_dir <arg>`	Directory	Specifies a log directory. If the `--log` option is specified, the default is to generate the log file in the current working directory (cwd). If no log directory is specified, the tool saves the files to `<cwd>/_x/logs`.
`--temp_dir <arg>`	Directory	Specifies a temp directory. If the `--save-temps` option is specified, the default is to create the temporary compilation and build files in the current working directory (cwd). If no temp directory is specified, the tool saves the files to `<cwd>/_x/reports`.
`--export_script`	N/A	Generates the Tcl script, `<kernel_name>.tcl`, used to execute Vivado HLS but halts before Vivado HLS starts. The expectation is for the script to be modified and used with the `--custom_script` option. Not supported for `-t sw_emu` with OpenCL kernels.
`--custom_script <arg>`	`<kernel_name>:<path to kernel Tcl file>`	Intended for use with the `<kernel_name>.tcl` file generated with `--export_script`. This option allows you to customize the Tcl file used to create the kernel and execute using the customize version of the script.
`<input file>`	OpenCL or C/C++ kernel source file, or Xilinx object file (`.xo`).	For Compile mode, the input file consist of OpenCL or C/C++ kernel source files. For Link mode, the input files consists of one or more Xilinx object files (`a.xo`).

Table 1: **XOCC Common Options (For Compile and Link Modes)** *(cont'd)*

Option	Valid Values	Description
`--user_ip_repo_paths <arg>`	`<directory>`	Specifies the directory location of the existing user IP repository. This value is prefixed to `ip_repo_paths`. Using this switch, specify one or more IP repository paths to be given highest priority by placing these paths at the beginning of the overall IP_REPO_PATHS property for the underlying Vivado project. IP definitions from any of these specified paths are used ahead of IP repositories from the hardware platform (`.dsa`) or from the Xilinx catalog. Multiple `--user_ip_repo_paths` can be specified. The following lists show the priority order in which IP definitions are found during SDx™ compilation flows (High to Low). Note that all of these entries can possibly include multiple directories in them. • For HW flow: 1. IP definitions from --user_ip_repo_paths switch 2. Kernel IP definitions (vpl --iprepo switch value) 3. IP definitions from DSA IP repo 4. IP cache dir from Install area (for example, `<SDxInstall>/SDx/2018.3/data/cache/`) 5. IP cache stored inside DSA 6. SDx specific Xilinx IPs from install area (for example, `<SDxInstall>/SDx/2018.3/data/ip/`) 7. General Xilinx IP catalog from install area (for example,`<SDxInstall>/Vivado/2018.3/data/ip/`) • For HW EMU flow: 1. IP definitions from --user_ip_repo_paths switch 2. User emulation ip repository (for example, `$::env(SDX_EM_REPO`)) 3. Kernel IP definitions (vpl --iprepo switch value) 4. IP cache dir from Install area (for example,`<SDxInstall>/SDx/2018.3/data/cache/`) 5. IP cache stored inside DSA 6. $::env(XILINX_SDX)/data/emulation/hw_em/ip_repo 7. $::env(XILINX_VIVADO)/data/emulation/hw_em/ip_repo 8. SDx Specific Xilinx IPs from install area (for example, `<SDxInstall>/SDx/2018.3/data/ip/`) 9. General Xilinx IP catalog from install area (for example, `<SDxInstall>/Vivado/2018.3/data/ip/`)

Table 1: **XOCC Common Options (For Compile and Link Modes)** *(cont'd)*

Option	Valid Values	Description
`--remote_ip_cache <arg>`	`<directory>`	Specifies remote IP cache directory for Vivado synthesis.
`--no_ip_cache`	N/A	Turns off IP cache for Vivado synthesis.
`--report_level <arg>`	Valid report levels: `0`, `1`,`2`, `estimate`. Example: `-R2`	These report levels have mappings kept in the optMap.xml file. You can override the installed optMap.xml to define custom report levels. • (Default) The `-R0` specification turns off all intermediate DCP generation during Vivado implementation. Turns on post route timing report generation. • The `-R1` specification turns on everything -R0 does, plus report_failfast pre-opt_design, report_failfast post-opt_design, and all intermediate DCP generation. • The `-R2` specification turns on everything -R1 does, plus it adds report_failfast post-route_design. • The `-Restimate` specification forces the Vivado HLS tools to generate a `design.xml` datafile if it does not exist, to generate an estimate report. This option is useful for software emulation target, when design.xml is not generated by default.
`--ini_file <arg>`	<path_to_file>	Reads in XP switches from file in `xocc.ini` format. This might be used multiple times for multiple files. These take priority over `xocc.ini` files found in default locations, but explicit `--xp` command line switches still take priority over those found in the specified file.
`--interactive <arg>`	`[ synth | impl ]`	`xocc` configures necessary environment and launches the Vivado toolset with either synthesis or implementation project.

XOCC Options for Compile Mode

Table 2: **XOCC Options for Compile Mode**

Option	Valid Values	Description
`--compile`	N/A	Required, but mutually exclusive with `--link`. Run `xocc` in compile mode,generate `.xo` file.
`--kernel <arg>`	Kernel to be compiled from the input `.cl` or `.c`/`.cpp` kernel source code.	Required for C/C++ kernels. Optional for OpenCL kernels. Compile/build only the specified kernel from the input file. Only one `-k` option is allowed per command. When an OpenCL kernel is compiled without the `-k` option, all the kernels in the input file are compiled.

Table 2: **XOCC Options for Compile Mode** *(cont'd)*

Option	Valid Values	Description
`--define <arg>`	Valid macro name and definition pair. `<name>=<definition>`	Pre-define name as a macro with definition. This option is passed to the `xocc` pre-processor.
`--include <arg>`	Directory name that includes required header files.	Adds the directory to the list of directories to be searched for header files. This option is passed to the SDAccel compiler preprocessor.
`--max_memory_ports <arg>`	`all | <kernel_name>`	Valid for OpenCL kernels. Sets the maximum memory ports for all kernels or for a given `<kernel name>`.
`--memory_port_data_width <arg>`	`all | <kernel_name>:<number>`	Valid for OpenCL kernels. Sets the memory port data width to the `<number>` for all kernels or for a given `<kernel name>`.
`--export_hls_project`		Specify a directory where a HLS project set-up script is exported
`--hls_export_mode`	Valid file types include: • `xo` • `tcl`	<file_type>:<file_path> Set an export mode from HLS with the path to an exported file.

XOCC Options for Link Mode

Table 3: **XOCC Options for Link Mode**

Option	Valid Values	Description
`--optimize <arg>`	Valid optimization levels: 0, 1, 2, 3, s, quick. `example: --optimize2` This option ONLY applies to Vivado. The compile step runs the C code through HLS and has no bearing.	These options control the default optimizations performed by the Vivado hardware synthesis engine. Familiarity with the Vivado tool suite is recommended to make better use of these settings. • 0: Default optimization. Reduce compilation time and make debugging produce the expected results. • 1: Optimize to reduce power consumption. This takes more time to compile the design. • 2: Optimize to increase kernel speed. This option increases both compilation time and the performance of the generated code. • 3: This is the highest level of optimization. This option provides the highest level performance in the generated code, but compilation time might increase considerably. • s: Optimize for size. This reduces the logic resources for the kernel • quick: Quick compilation for fast run time. This might result in reduced hardware performance, and a greater use of resources in the hardware implementation.
`user_board_repo_paths`		Specify existing user board repository for DIMM board files. This value will be appended to board_part_repo_paths
`--clkid`		Select an index available from SoC platform. Each index has a different default. clock frequency
`--board_connection`		<DIMM_connector:vbnv_of_DIMM_board> Specify a dual in-line memory module (DIMM) board file for each DIMM connector slot.
`-l / --link`	N/A	Required, but mutually exclusive with `--compile`. Run `xocc` in link mode. Link `.xo` input files, generate `.xclbin` file.

Table 3: **XOCC Options for Link Mode** *(cont'd)*

Option	Valid Values	Description
`--nk <arg>`	For example: `foo:2` `<kernel_name:number(:compute_unit_name1.compute_unit_name2...)>` For example: `foo:3:fooA.fooB.fooC`	This option instantiates the specified number of compute units for the given kernel in the <kernel_name:number>`.xclbin` file. The compute unit (CU) name is optional. If the CU name is not specified, the instances of the kernel are simply numbered: `kernel_name_1`, `kernel_name_2`, and so forth. <By default, the XOCC instantiates one compute unit for each kernel. Optional in link mode. Not applicable in compile mode.
`--jobs <arg>`	Number of parallel jobs.	Optional. This option allows detailed control of the Vivado Design Suite used to implement the FPGA hardware. Familiarity with the Vivado Design Suite is recommended to make the most use of this option. Specifies the number of parallel jobs to be passed to the Vivado Design Suite for implementation. Increasing the number of jobs allows the hardware implementation step to spawn more parallel processes and complete faster.
`--lsf <arg>`	`bsub` command line to pass to LSF cluster. This argument is required for use with `--lsf`.	Optional. Use IBM Platform Load Sharing Facility (LSF) for Vivado implementation and synthesis. For example: `--lsf '{bsub -R "select[type=X86_64]" -N -q medium}'`
`--reuse_impl`	<Implemented DCP>	Imports an implemented DCP and runs only the XCLBIN packaging.

Table 3: **XOCC Options for Link Mode** *(cont'd)*

Option	Valid Values	Description
`--dk <arg>`	`<[protocol|chipscope|list_ports]:<compute_unit_name>:<interface_name>>` Where: • `<interface_name>` is optional. If not specified, all ports are expected to be analyzed. • The `chipscope` option requires the explicit name of the compute unit to be provided for the `<compute_unit_name>` and `<interface_name>`. • The `protocol` option can accept a special keyword "all" for `<compute_unit_name>` and for `<interface_name>`. The `chipscope` option can not accept this keyword "all." • The `list_ports` option shows a list of valid compute units and port combinations in the current design.	Enables debug IP core insertion. Allows you to specify which compute unit interfaces to monitor with chipscope. This is useful for hardware debugging. The System ILA debug core provides transaction level visibility into an accelerated kernel or function running on hardware. AXI traffic of interest can also be captured and viewed using the System ILA core. The `--dk` option allows you to attach System ILA cores at the interfaces to the kernels for debugging and performance monitoring purposes.
`--slr`	`<compute_unit_name>:<SLR_NUM>` Where: • `<compute_unit_name>` is the name of the compute unit as specified in the `--nk` option. Generally this will be <kernel_name> unless a different name was specified. • `<SLR_NUM>` is the SLR number to which the CU is assigned. For example, SLR0, SLR1. • The option must be repeated for each kernel or CU being assigned.	Use `--slr` to assign a Compute Unit (CU) to an SLR.. For example, to assign CU `vadd_2` to SLR2, and CU `fft_1` to SLR1, you would use the following: `--slr vadd_2:SLR2` `--slr fft_1:SLR1` **IMPORTANT!** *If you use `--slr` to assign the kernel placement, then you must also use `--sp` to assign memory access for the kernel.*

Table 3: **XOCC Options for Link Mode** *(cont'd)*

Option	Valid Values	Description
`--sp <arg>` Starting in release 2018.3, memory resource are specified using vector formatting with the resource item enclosed in square brackets (that is, [..]). For example, the DDR memory resource names of a device with four (4) DDR banks are specified at DDR[0], DDR[1], DDR[2], and DR[3]. PLRAM is specified in a similar fashion. While release 2018.3 supports the legacy sptag names (that is, bank<n>) for platforms available in 2018.2.xdf and any associated updates in 2018.3, this support will be deprecated in subsequent releases. All new platforms in 2018.3, however, do not support legacy sptag names and require the vector syntax format.	`<compute_unit_name>.<kernel_interface_name>:<sptag[min:max]>` Where: • `<compute_unit_name>` is the name of the compute unit as specified in the `--nk` option. Generally this will be <kernel_name> unless a different name was specified. • `<kernel_interface_name>` is the name of the function argument for the kernel, or compute unit port. • `<sptag>` represents a memory resource name from the target platform. Valid `<sptag>` names include DDR, PLRAM, and HBM. • `[min:max]` enables the use of a range of memory, such as DDR[0:2]. A single index is also supported: DDR[2]. **TIP:** *The supported <sptag> and range of memory resources for a target platform can be obtained using the `platforminfo` command. Refer to Chapter 7: platforminfo Utility for more information.*	Use the system port option (`--sp` to connect a kernel interface, or CU, to a specific type and range of memory resources. Multiple `--sp` options can be specified to map each of a kernel's interfaces to a particular memory resource. In absence of `--sp` option, the XOCC linker will attempt to connect the kernel interfaces to a memory resource in the same SLR The following example maps the input arguments (A and B) for the specified CU of the VADD kernel to DDR[0:3], and writes the output argument (C) to PLRAM[2]: `--sp vadd_1.A:DDR[0:3]` `--sp vadd_1.B:DDR[0:3]` `--sp vadd_1.C:PLRAM[2]`
`--sys_config <arg>`	Valid value is `ocl`	Specifies a system configuration setting for SoC platforms. Valid value is `ocl`. In platform spfm file, the configuration section should contain `sdx:runtimes = "ocl."`

XP Parameters

When compiling or linking, fine grain control over the hardware generated by the SDAccel tools and the hardware emulation process can be specified by using the `--xp` switch.

The `--xp` switch is paired with parameters to configure the Vivado tools. For instance, the switch `--xp` can configure optimization, placement and timing, or set up emulation and compile options. Specific examples of these parameters include setting the clock margin, specifying the depth of FIFOs used in the kernel dataflow region, and specifying the number of outstanding writes and reads to buffer on the kernel AXI interface.

IMPORTANT! *Familiarity with the Vivado Design Suite is required to make the most use of these parameters. See Vivado Design Suite User Guide: High-Level Synthesis (UG902) and Vivado Design Suite User Guide: Implementation (UG904) for more information.*

Parameters are specified as `parm:<param_name>=<value>`. For example:

```
xocc --xp param:compiler.enableDSAIntegrityCheck=true
-xp param:prop:kernel.foo.kernel_flags="-std=c++0x"
```

You can specify the `--xp` command option multiple times in a single `xocc` invocation or specify the value(s) in an `xocc.ini` file with each option specified on a separate line without `--xp` switch.

```
param:prop:solution.device_repo_paths=../dsa
param:compiler.preserveHlsOutput=1
```

Upon invocation, `xocc` first looks for an `xocc.ini` file in the `$HOME/.Xilinx/sdx` directory. If the file does not exist, then `xocc` looks for it in the current working directory. If the same `--xp` parameter value is specified in both the command line and `xocc.ini` file, the command line value is used.

The following table lists a sample of the `--xp` parameters and their values.

Table 4: **XP Parameter Options**

Parameter Name	Valid Values	Description
`param:compiler.acceleratorBinaryContent`	Type: String Default Value: `<empty>`	Content to insert in `xclbin`. Valid options are `bitstream` and `dcp`.
`param:compiler.errorOnHoldViolation`	Type: Boolean Default Value: TRUE	Error out if there is hold violation.
`param:compiler.maxComputeUnits`	Type: Int Default Value: `-1`	Maximum compute units allowed in the system. Any positive value will overwrite the `numComputeUnits` setting in the hardware platform (`.dsa`). The default value of -1 preserves the setting in the DSA.
`param:hw_em.compiledLibs`	Type: String Default Value: `<empty>`	Uses mentioned `clibs` for the specified simulator.

Table 4: **XP Parameter Options** *(cont'd)*

Parameter Name	Valid Values	Description
`param:hw_em.platformPath` <absolute_path_of_custom_platform_directory>	Type: String Default Value: `<empty>`	Specifies the path to the custom platform directory. The `<platformPath>` directory should meet the following requirements to be used in platform creation: • The directory should contain a subdirectory called `ip_repo`. • The directory should contain a subdirectory called `scripts` and this `scripts` directory should contain a `hw_em_util.tcl` file. The `hw_em_util.tcl` file should have following two procedures defined in it: ◦ hw_em_util::add_base_platform ◦ hw_em_util::generate_simulation_scripts_and_compile
`param:hw_em.enableProtocolChecker`	Type: Boolean Default Value: FALSE	Enables the lightweight AXI protocol checker (lapc) during HW emulation. This is used to confirm the accuracy of any AXI interfaces in the design.
`param:compiler.xclDataflowFifoDepth`	Type: Int Default Value: `-1`	Specifies the depth of FIFOs used in kernel data flow region.
`param:compiler.interfaceWrOutstanding`	Type: Int Range Default Value: `0`	Specifies how many outstanding writes to buffer are on the kernel AXI interface. Values are 1 through 256.
`param:compiler.interfaceRdOutstanding`	Type: Int Range Default Value: `0`	Specifies how many outstanding reads to buffer are on the kernel AXI interface. Values are 1 through 256.
`param:compiler.interfaceWrBurstLen`	Type: Int Range Default Value: `0`	Specifies the expected length of AXI write bursts on the kernel AXI interface. This is used with option `compiler.interfaceWrOutstanding` to determine the hardware buffer sizes. Values are 1 through 256.
`param:compiler.interfaceRdBurstLen`	Type: Int Range Default Value: `0`	Specifies the expected length of AXI read bursts on the kernel AXI interface. This is used with option `compiler.interfaceRdOutstanding` to determine the hardware buffer sizes. Values are 1 through 256.
`misc:map_connect=<type>.kernel.<kernel_name>.<kernel_AXI_interface>.core.OCL_REGION_0.<dest_port>`	Type: String Default Value: `<empty>`	Used to map AXI interfaces from a kernel to DDR memory banks. • `<type>` is add or remove. • `<kernel_name>` is the name of the kernel. • `<dest_port>` is a DDR memory bank M00_AXI, M01_AXI, M02_AXI, or M03_AXI. This option is available only for DSA 4.x and earlier and deprecated for DSA 5.x and later. Use system ports using the `--sp` option documented in XOCC Options for Link Mode.

Table 4: **XP Parameter Options** *(cont'd)*

Parameter Name	Valid Values	Description
`prop:kernel.<kernel_name>.kernel_flags`	Type: String Default Value: `<empty>`	Sets specific compile flags on the kernel `<kernel_name>`.
`prop:solution.device_repo_path`	Type: String Default Value: `<empty>`	Specifies the path to a repository of hardware platforms. The `--platform` option with full path to the `.xpfm` platform file should be used instead.
`prop:solution.hls_pre_tcl`	Type: String Default Value: `<empty>`	Specifies the path to a Vivado HLS Tcl file, which is executed before the C code is synthesized. This allows Vivado HLS configuration settings to be applied prior to synthesis.
`prop:solution.hls_post_tcl`	Type: String Default Value: `<empty>`	Specifies the path to a Vivado HLS Tcl file, which is executed after the C code is synthesized.
`prop:solution.kernel_compiler_margin`	Type: Float Default Value: 12.5% of the kernel clock period.	The clock margin (in ns) for the kernel. This value is subtracted from the kernel clock period prior to synthesis to provide some margin for P&R delays.
`vivado_prop:<object_type>.<object_name>.<prop_name>`	Type: Various Default Value: Various	This allows you to specify any property used in the Vivado hardware compilation flow. `<object_type>` is `run|fileset|file|project`. The `<object_name>` and `<prop_name>` values are described in *Vivado Design Suite Properties Reference Guide* (UG912). Examples: `vivado_prop:run.impl_1.{STEPS.PLACE_DESIGN.ARGS.MORE OPTIONS}={-fanout_opt}` `vivado_prop:fileset.current.top=foo` If `<object_type>` is set to `file`, `current` is not supported. If `<object type>` is set to `run`, the special value of `__KERNEL__` can be used to specify run optimization settings for ALL kernels, instead of the need to specify them one by one.

Using the Message Rule File

XOCC executes various Xilinx tools during kernel compilation. These tools generate many messages that provide compilation status to you. These messages might or might not be relevant to you depending on your focus and design phases. A Message Rule file can be used to better manage these messages. It provides commands to promote important messages to the terminal or suppress unimportant ones. This helps you better understand the kernel compilation result and explore methods to optimize the kernel.

The Message Rule file (`.mrf`) is a text file consisting of comments and supported commands. Only one command is allowed on each line.

Comment

Any line with "`#`" as the first non-white space character is a comment.

Supported Commands

By default, `xocc` recursively scans the entire working directory and promotes all error messages to the `xocc` output. The `promote` and `suppress` commands below provide more control on the `xocc` output.

- `promote`: This command indicates that matching messages should be promoted to the `xocc` output.
- `suppress`: This command indicates that matching messages should be suppressed or filtered from the `xocc` output. Note that errors cannot be suppressed.

Command Options

The Message Rule file can have multiple `promote` and `suppress` commands. Each command can have one and only one of the options below. The options are case-sensitive.

- `-id [<message_id>]`: All messages matching the specified message ID are promoted or suppressed. The message ID is in format of nnn-mmm. As an example, the following is a warning message from HLS. The message ID in this case is 204-68.

  ```
  WARNING: [XOCC 204-68] Unable to enforce a carried dependence constraint
  (II = 1, distance = 1, offset = 1)
  between bus request on port 'gmem'
  (/matrix_multiply_cl_kernel/mmult1.cl:57) and bus request on port 'gmem'-
  severity [severity_level]
  ```

  For example, to suppress messages with message ID 204-68, specify the following: `suppress -id 204-68`.

- `-severity [<severity_level>]`: The following are valid values for the severity level. All messages matching the specified severity level will be promoted or suppressed.

- `info`
- `warning`
- `critical_warning`

  For example, to promote messages with severity of 'critical-warning', specify the following: `promote -serverity critical_warning`.

Precedence of Message Rules

The `suppress` rules take precedence over `promote` rules. If the same message ID or severity level is passed to both `promote` and `suppress` commands in the Message Rule file, the matching messages are suppressed and not displayed.

Example of Message Rule File

The following is an example of a valid Message Rule file:

```
# promote all warning, critical warning
promote -severity warning
promote -severity critical_warning
# suppress the critical warning message with id 19-2342
suppress -id 19-2342
```

SDSCC/SDS++ Compiler Commands

This section describes the SDSoC™ `sds++` compiler commands and options.

Note: `sdscc` and `sds++` are compilers based on GCC, and therefore support many standard GCC options which are not documented here. For information refer to the GCC Option Index.

Compiler Commands

```
sdscc - SDSoC C compiler
```

```
sds++ - SDSoC C++ compiler
```

Command Synopsis

```
sdscc | sds++ [hardware_function_options] [system_options]
           [performance_estimation_options]
           [options_passed_through_to_cross_compiler]
           [-sds-pf platform_name] [-sds-pf-info platform_name]
           [-sds-pf-list] [-sds-sys-config configuration_name]
           [-sds-proc processor_name] [-target-os os_name]
           [-debug-xrf] [-debug-xrf-cc compiler]
           [-sds-pf-path path] [-sds-image image_name]
           [-verbose] [-version] [--help] [files]
```

Hardware Function Options

```
[-sds-hw function_name source_file [-clkid clock_id_number]
[-files hls_file_list] [-hls-target boolean_value] [-hls-target-flags
"target_options"] [-hls-tcl hls_tcl_directives_file]
[-shared-aximm] -sds-end]*
```

For detail on these commands, see Hardware Function Options.

Performance Estimation Options

```
[[-perf-funcs function_name_list -perf-root function_name] |
[-perf-est data_file][-perf-est-hw-only]]
```

For detail on these commands, see SDSCC/SDS++ Performance Estimation Flow Options.

System Options

```
[[-ac function_name:clock_id_number]*[-apm] [-bsp-config-file mss_file]
[-bsp-config-merge-file mss_file][-disable-ip-cache] [-dm-sharing <0-3>] [-
dmclkid clock_id_number]
[-emulation mode] [-impl-strategy <strategy>]
[-instrument-stub] [-maxthreads number] [-mno-bitstream][-mno-boot-files] [-
rebuild-hardware]
[-remote-ip-cache cache_directory]
[-synth-strategy <strategy>] [-trace] [-trace-buffer depth] [-trace-no-sw]
[-maxjobs <number>]
[-sdcard <data_directory>][-vpl-ini ini_file] [-xp parameter_value]]
```

For details on these commands, see System Options.

The `sds++` compiler compiles and links C/C++ source files into an application-specific hardware/software system-on-a-chip (SoC) implemented on a Zynq®-7000 SoC or Zynq® UltraScale+™ MPSoC device.

The command usage and options are identical for `sdscc` and `sds++`. Options not recognized by `sds++` are passed to the Arm® cross-compiler. Compiler options within an `-sds-hw ... -sds-end` clause are ignored for the `-c foo.c` option when `foo.c` is not the file containing the specified hardware function.

When linking the application ELF, `sds++` creates and implements the hardware system. It also generates an SD card image containing the ELF and boot files required to initialize the hardware system, configures the programmable logic, and runs the target operating system.

When linking application ELF files for non-Linux targets, for example Standalone or FreeRTOS, default linker scripts found in the folder `<install_path>/platforms/<platform_name>` are used. If a user-defined linker script is required, it can be specified using the `-Wl,-T -Wl,<path_to_linker_script>` linker option.

When building a system containing no functions marked for hardware implementation, `sds++` uses pre-built hardware when available for the target platform.

Report and log files are found in the `_sds/reports` folder.

When running Linux applications that use shared libraries, the libraries must be contained in the root file system or SD card and the path to the libraries added to the `LD_LIBRARY_PATH` environment variable.

Optional PL Configuration After Linux Boot

When `sds++` creates a bitstream `.bin` file in the `sd_card` folder, it can be used to configure the PL after booting Linux and before running the application ELF. The embedded Linux command used is `cat bin_file > /dev/xdevcfg`.

General Options

The following command line options are applicable for any `sds++` invocation or your display information.

Table 5: **General Options**

Option	Valid Values	Description
`-sds-pf <platform_name>`	`<platform_name>`	Specifies the target platform that defines the base system hardware and software, including operation system and boot files. The `<platform_name>` can be the name of a platform in the SDSoC environment installation or a file path to a folder containing platform files with the last component of the path matching the platform name. The platform defines the base hardware and software, including operation system and boot files. Use this option when compiling accelerator source files and when linking the ELF file. Use the `-sds-pf-list` option to list available platforms.
`-sds-pf-info <platform_name>`	`<platform_name>`	Displays general information about a platform. Use the `-sds-pf-list` option to list available platforms. The information displayed includes available system configurations that can be specified with the `-sds-sys-config` system_configuration option. `<platform_name>` can be the name of a platform in the SDSoC environment installation or a file path to a folder containing platform files.
`-sds-pf-list`	N/A	Displays a list of available platforms and exit (if no other options are specified). The information displayed includes available system configurations that can be specified with the `-sds-sys-config` system_configuration option.
`-sds-sys-config <configuration_name>`	`<configuration_name>`	Specifies the system configuration that defines the software platform used, which includes the target operating system and other settings. The `-sds-pf-list` and `-sds-pf-info` options can be used to list the available system configurations for a platform. When the `-sds-sys-config` option is used, do not specify the `-target-os` option. If the `-sds-sys-config` option is not specified, the default system configuration is used. `<configuration_name>` can be any of the available system configurations for a platform.
`-sds-proc <processor_name>`	`<processor_name>`	Specifies the processor name to use with the system configuration defined by the `-sds-sys-config` option. A system configuration normally specifies a target CPU and this option is not required. `<processor_name>` specifies the target CPU to use.

Table 5: **General Options** *(cont'd)*

Option	Valid Values	Description
`-sds-pf-path <path>`	`<path>`	Specifies a search path for platforms. The specified path can contain one or more sub-folders, each of which is a platform folder. `<path>` is a search path for platforms.
`-sds-image <image_name>`	`<image_name>`	Used with the `-sds-sys-config` option, this specifies the SD card image to use. If this option is not specified, the default image is used. `<image_name>` specifies the SD card image to use.
`-target-os <os_name>`	`<linux | standalone | freertos>`	Specifies the target operating system. The selected OS determines the compiler toolchain used and includes file and library paths added by `sds++`. For a list of valid `os_name` options, use the command `sdscc -sds-pf-info <plat_name>`. If the `-sds-sys-config` system_configuration option is specified, do not specify the `-target-os` option, because a system configuration itself defines a target operating system. If you do not specify the `-sds-sys-config` but do specify the `-target-os` option, SDSoC searches for a system configuration with an OS that matches the one specified by `-target-os`.
`-debug-xrf`	N/A	Creates Vivado® HLS debug cross reference information when compiling functions for hardware, adding the config_debug Tcl command.
`-debug-xrf-cc <compiler>`	`<compiler>`	Specifies the compiler executable used to compile accelerator source files for debug cross reference symbols (defaults to `xcpp` if not specified). If the path to the executable is not defined, use the PATH from the environment. If used, specify `-debug-xrf-cc` for both compile and link command lines.
`-verbose`	N/A	Prints verbose output to STDOUT.
`-version`	N/A	Prints the `sds++` version information to STDOUT.
`--help`	N/A	Prints command line help information. Note that two consecutive hyphen or dash characters `-` are used.

The following command line options are applicable only to `sds++` invocations used to compile a source file.

Hardware Function Options

Hardware function options provide a means to consolidate `sdscc`/`sds++` options within a `Makefile` to simplify command line calls and make minimal modifications to a pre-existing `Makefile`.

The `-sds-hw` and `-sds-end` options are used in pairs:

- The `-sds-hw` option begins the description of a single function being moved into hardware.
- `-sds-end` option terminates the list of configuration details for that function.

For the next function moved into hardware, there is another pair with `-sds-hw` as the start of the configuration and `-sds-end` as the terminator.

The `Makefile` fragment below illustrates the use of `-sds-hw` blocks to collect all options in the `SDSFLAGS` `Makefile` variable and to replace an original definition of `CC` with `sds++ ${SDSFLAGS}`. Thus the original `Makefile` for an application can be converted to an `sds++` compiler `Makefile` with minimal changes.

```
APPSOURCES = add.cpp main.cpp
EXECUTABLE = add.elf

CROSS_COMPILE = arm-xilinx-linux-gnueabi-
AR = ${CROSS_COMPILE}ar
LD = ${CROSS_COMPILE}ld
#CC = ${CROSS_COMPILE}g++
PLATFORM = zc702
SDSFLAGS = -sds-pf ${PLATFORM} \
           -sds-hw add add.cpp -clkid 1 -sds-end \
           -dmclkid 2
CC = sds++ ${SDSFLAGS}

INCDIRS = -I..
LDDIRS =
LDLIBS =
CFLAGS = -Wall -g -c ${INCDIRS}
LDFLAGS = -g ${LDDIRS} ${LDLIBS}

SOURCES := $(patsubst %,../%,$(APPSOURCES))
OBJECTS := $(APPSOURCES:.cpp=.o)

.PHONY: all

all: ${EXECUTABLE}

${EXECUTABLE}: ${OBJECTS}
 ${CC} ${OBJECTS} -o $@ ${LDFLAGS}

%.o: ../%.cpp
 ${CC} ${CFLAGS} $<
```

Table 6: **Hardware Function Options**

Option	Valid Values	Description
`-sds-hw function_name source_file`	N/A	An `sds++` command line can include zero or more `-sds-hw` blocks. Each block is associated with a top-level hardware function specified as the first argument and its containing source file specified as the second argument. If the file name associated with an `-sds-hw` block matches the source file to be compiled, the options are applied. Options outside of `-sds-hw` blocks are applied where applicable. When using the xfOpenCV library, the `function_name` is the template function instantiation enclosed in double quotes, for example `"auCanny<1080,1920,0,0,3,2,1,1,1>"`, and the file is the source file containing the template function instantiation, for example `au_canny_tb.cpp`.
`-clkid <n>`	`<n>` has one of the values listed in the Appendix A: Clock ID Values by Platform table.	Sets the accelerator clock ID to `<n>`, where `<n>` has one of the values listed in the Appendix A: Clock ID Values by Platform table. (You can use the command `sds++ -sds-pf-info platform_name` to display the information about a platform.) If the `clkid` option is not specified, the default value for the platform is used. Use the command `sds++ -sds-pf-list` to list available platforms and settings.
`-files file_list`	`file_list` is a list of one or more files required to compile the current top-level function into hardware using Vivado HLS.	Specifies a comma-separated list (without white space) of one or more files required to compile the current top-level function into hardware using Vivado HLS. If any of these files contain source code that is not used by HLS but is required to produce the application executable, they must be compiled separately to create object files (`.o`), and linked with other object files during the link phase. When using the xfOpenCV library, the `-files` option specifies the path to the source file containing the function template definition, for example `au_canny.hpp`.
`-hls-target boolean_value`	`boolean_value` is `0|1`	When set to 1, in Vivado HLS `add_files` commands, insert `-target` and Arm GNU toolchain include options in addition to `-m32` or `-m64` options. When set to 0, insert `-m32` or `-m64` options. Use this option if the default behavior results in target dependent compilation errors. When specified outside of `-sds-hw`/`-sds-end` blocks, the option applies to all hardware functions.
`-hls-target-flags "target_options"`	`"target_options"` are options in the Vivado HLS `add_files` command.	Specifies a list of target options to use in place of automatically inserted options in the Vivado HLS `add_files` command, for example `-m32` or `-m64`. The options must be enclosed in quotes so they will not be interpreted as compiler options.

Table 6: **Hardware Function Options** *(cont'd)*

Option	Valid Values	Description
`-hls-tcl hls_tcl_directives_file`	N/A	When using the Vivado HLS tool to synthesize the hardware accelerator, source the specified Tcl file containing HLS directives. During HLS synthesis, `sds++` creates a `run.tcl` file used to drive the Vivado HLS tool. In this Tcl file, the following commands are inserted: `# synthesis directives` `create_clock -period <clock_period>` `set_clock_uncertainty 27.0%` `config_rtl -reset_level low` `source <sdsoc_generated_tcl_directives_file>` `# end synthesis directives` If the `-hls-tcl` option is used, the user-defined Tcl file is sourced after the synthesis directives generated by the SDSoC environment.
`-shared-aximm`	N/A	Shares AXIMM ports instead of enabling multiple ports.
`-sds-end`	N/A	Specifies the end of the `-sds-hw` options for the specified `function_name`.

Clock ID Values by Platform

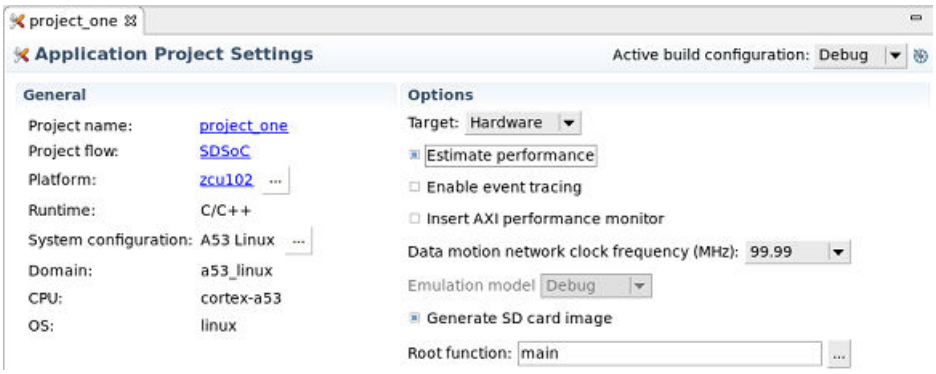
For a list of clock ID values by platform, see Appendix A: Clock ID Values by Platform.

SDSCC/SDS++ Performance Estimation Flow Options

A full bitstream compile can take much more time than a software compile, so the `sdscc/sds++` (referred to as `sds++`) applications provide performance estimation options to compute the estimated runtime improvement for a set of hardware function calls.

In the Application Project Settings pane, to invoke the estimator, select the **Estimate Performance** check box. This enables performance estimation for the current build configuration and builds the project.

Figure 1: **Setting Estimate Performance in Application Project Settings**



Estimating the speed-up is a two phase process:

1. The SDSoC environment compiles the hardware functions and generates the system. Instead of synthesizing the system to bitstream, the `sds++` computes an estimate of the performance based on estimated latencies for the hardware functions and data transfer time estimates for the callers of hardware functions.
2. In the generated Performance Report, to determine a performance baseline and the performance estimate, select **Click Here** to run an instrumented version of the software on the target .

See the *SDSoC Environment Getting Started Tutorial* (UG1028) for a tutorial on how to use the Performance Report.

You can also generate a performance estimate from the command line. As a first pass to gather data about software runtime, use the `-perf-funcs` option to specify functions to profile and `-perf-root` to specify the root function encompassing calls to the profiled functions.

The `sds++` system compiler then automatically instruments these functions to collect runtime data when the application is run on a board. When you run an instrumented application on the target, the program creates a file on the SD card called `swdata.xml`, which contains the runtime performance data for the run.

Copy the `swdata.xml` to the host, and run a build that estimates the performance gain on a per hardware function caller basis and for the top-level function specified by the `-perf-root` function in the first pass run. Use the `-perf-est` option to specify `swdata.xml` as input data for this build.

The following table specifies the `sds++` system compiler options normally used to build an application.

Table 7: **Commonly used sds++ options**

Option	Description
`-perf-funcs function_name_list`	Specifies a comma separated list of all functions to be profiled in the instrumented software application.

Table 7: **Commonly used sds++ options** *(cont'd)*

Option	Description
`-perf-root function_name`	Specifies the root function encompassing all calls to the profiled functions. The default is the function main.
`-perf-est data_file`	Specifies the file containing runtime data generated by the instrumented software application when run on the target. Estimate performance gains for hardware accelerated functions. The default name for this file is `swdata.xml`.
`-perf-est-hw-only`	Runs the estimation flow without running the first pass to collect software run data. Using this option provides hardware latency and resource estimates without providing a comparison against baseline.

CAUTION! *After running the `sd_card` image on the board for collecting profile data, type `cd /; sync; umount /mnt;`. This ensures that the `swdata.xml` file is written out to the SD card.*

Compiler Macros

Predefined macros allow you to guard code with `#ifdef` and `#ifndef` preprocessor statements. The macro names begin and end with two underscore characters '_'. The `__SDSCC__` macro is defined whenever `sdscc` or `sds++` (referred to collectively as `sds++`) is used to compile source files. It can be used to guard code depending on whether it is compiled by `sds++` or another compiler, for example `GCC`.

When `sds++` compiles source files targeted for hardware acceleration using Vivado HLS, the `__SDSVHLS__` macro is defined to be used to guard code depending on whether high-level synthesis is run or not.

The code fragment below illustrates the use of the `__SDSCC__` macro to use the `sds_alloc()` and `sds_free()` functions when compiling source code with `sds++`, `malloc()`, and `free()` when using other compilers.

```
#ifdef __SDSCC__
#include <stdlib.h>
#include "sds_lib.h"
#define malloc(x) (sds_alloc(x))
#define free(x) (sds_free(x))
#endif
```

In the example below, the `__SDSVHLS__` macro is used to guard code in a function definition that differs depending on whether it is used by Vivado HLS to generate hardware or used in a software implementation.

```
#ifdef __SDSVHLS__
void mmult(ap_axiu<32,1,1,1> A[A_NROWS*A_NCOLS],
           ap_axiu<32,1,1,1> B[A_NCOLS*B_NCOLS],
           ap_axiu<32,1,1,1> C[A_NROWS*B_NCOLS])
#else
void mmult(float A[A_NROWS*A_NCOLS],
           float B[A_NCOLS*B_NCOLS],
           float C[A_NROWS*B_NCOLS])
#endif
```

In addition, the macro, `HLS_NO_XIL_FPO_LIB`, is defined prior to the include option for Vivado HLS headers and is visible to Vivado HLS, SDSoC analysis tools, and target cross-compilers. This macro disables the use of bit-accurate, floating-point simulation models, instead using the faster (although not bit-accurate) implementation from your local system. Bit-accurate simulation models are not provided for Zynq-7000 SoC and Zynq UltraScale+ MPSoC Arm targets.

System Options

Table 8: **System Options**

Option	Description
`-ac <function_name>:<clock_id_number>`	Uses the specified clock ID number for an RTL accelerator function in a C-Callable IP library instead of the default clock ID.
`-apm`	Inserts an AXIAXI Performance Monitor IP block to monitor all generated hardware/software interfaces. Within the SDSoC development environment, in the Debug perspective, you can activate the APM prior to running your application by clicking the **Start** button within the Performance Counters View. See the *SDSoC Environment Getting Started Tutorial* (UG1028) for more information.
`-bsp-config-file <mss_file>`	Specifies the path to a board support package (BSP) configuration file (`.mss`) to use instead of an automatically-generated file for a bare-metal based target OS, for example Standalone or FreeRTOS. When using this option, also add an include option specifying the path to your BSP header files: `-I</path/to/includes>`
`-bsp-config-merge-file <mss_file>`	Specifies the path to a board support package (BSP) configuration file (`.mss`) to use for the base platform and merge using hardware information from the final design to create a BSP configuration file contain user settings for the base platform plus settings for hardware added to the base platform; for example, DMA drivers. This merged BSP configuration file is used instead of an automatically generated file for a bare-metal based target OS, for example, Standalone or FreeRTOS. When using this option, add an include option specifying the path to your BSP header files: `-I</path/to/includes>`.

Table 8: **System Options** *(cont'd)*

Option	Description
`-debug-port function:argument`	Specifies a function and argument to monitor using a System ILA module. Multiple `-debug-port` options can be specified, instantiating a System ILA as needed. To specify the instance name and port to monitor instead, use the `-dk` option (`sds++` maps `-debug-port` to `-dk` options).
`-disable-ip-cache`	Do not use a cache of pre-synthesized IP cores. The use of IP caching reduces the overall build time by eliminating the synthesis step for static IP cores. If the resources required to implement the hardware system exceeds available resources by a small amount, the `-disable-ip-cache` option forces `sds++` to synthesize all IP cores in the context of the design and might reduce resource usage enough to enable implementation.
`-dmclkid <n>`	Sets the data motion network clock ID to `<n>`, where `<n>` has one of the values listed in Appendix A: Clock ID Values by Platform. You can use the command `sds++ -sds-pf-info platform_name` to display the information about the platform. If the `dmclkid` option is not specified, the default value for the platform is used. Use the command `sds++ -sds-pf-list` to list available platforms and settings.
`-dk chipscope:instance:port`	Specifies an instance name and port to monitor using a System ILA module. Multiple `-dk` options can be specified, instantiating a System ILA as needed. For special cases, use the option `--xp param:compiler.userPostSysLinkTcl=<file>` to specify a Tcl file containing VivadoIP integrator Tcl commands to post-process the System ILA in the block diagram after system linking and before synthesis. **Note:** `--dk` is also accepted.
`-dk list_ports`	Lists available instance and port names for System ILA insertion. This option can only be specified when linking the design and you can specify `-mno-bitstream` to exit, review `<pwd>/_sds/p0/dk_list_ports.txt`, and update the command line to create the bitstream. **Note:** `--dk` is also accepted.
`-dm-sharing <n>`	The `-dm-sharing <n>` option enables exploration of data mover sharing capabilities if the initial schedule can be relaxed. The level of sharing defaults to 0 (low) if not specified. Other values are 1 (medium), 2 (high), and 3 (maximum – schedule can be relaxed infinitely). For example, to enable maximum data mover sharing, add the `sds++ -dm-sharing 3` option.
`-emulation <mode>`	Generates files required to run emulation of the system using QEMU for the processing subsystem and the Vivado Logic Simulator for the programmable logic. This only works on boards that enable this flow (currently Xilinx base platforms only). The `<mode>` specifies the type of simulation models created for the PL, `debug`, or `optimized`. In the same directory that you ran `sds++`, type the `sdsoc_emulator` command to run the emulation in the current shell.

Table 8: **System Options** *(cont'd)*

Option	Description
`-impl-strategy <strategy_name>`	Specifies the Vivado implementation strategy name to use instead of the default strategy, for example Performance_Explore. The strategy name can be found in the Vivado Implementation Settings dialog box in the Strategy menu, and the strategies are described in this link in the *Vivado Design Suite User Guide: Implementation* (UG904). **Note:** When creating the Tcl file for synthesis and implementation, this command is added: `set_property strategy <strategy_name> [get_runs impl_1]`.
`-instrument-stub`	The `-instrument-stub` option instruments the generated hardware function stubs with calls to the counter function `sds_clock_counter()`. When a hardware function stub is instrumented, the time required to call send and receive functions, as well as the time spent for waits, is displayed for each call to the function.
`-maxjobs <n>`	The `-maxjobs <n>` option specifies the maximum number of jobs used for Vivado synthesis. The default is the number of cores divided by 2.
`-maxthreads <n>`	The `-maxthreads <n>` option specifies the number of threads used in multithreading to speed up certain tasks, including Vivado placement and routing. The number of threads can be an integer from 1 to 8. The default value is 4, but the tools do not use more threads than the number of cores on the machine. Also, a general limit based on the OS applies to all tasks.
`-mno-bitstream`	Do not generate the bitstream for the design used to configure the programmable logic (PL). Normally a bitstream is generated by running the Vivado implementation feature, which can be time-consuming with run times ranging from minutes to hours depending on the size and complexity of the design. This option can be used to disable this step when iterating over flows that do not impact the hardware generation. The application ELF is compiled before bitstream generation.
`-mno-boot-files`	Do not generate the SD card image in the folder `sd_card`. This folder includes your application ELF and files required to boot the device and bring up the specified OS. This option disables the creation of the `sd_card` folder in case you would like to preserve an earlier version of this folder.
`-rebuild-hardware`	When building a software-only design with no functions mapped to hardware, `sds++` uses a pre-built bitstream if available within the platform, but use this option to force a full system build.
`-remote-ip-cache <cache_directory>`	Specifies the path to a directory used for IP caching for Vivado synthesis. The use of an IP cache can reduce the amount of time required for logic synthesis for subsequent runs. The option `--remote_ip_cache` is also accepted.
`-sdcard <data_directory>`	Specifies an optional directory containing additional files to include in the SD card image.
`-synth-strategy <strategy_name>`	Specifies the Vivado synthesis strategy name to use instead of the default strategy (for example, Flow_RuntimeOptimized). The strategy name can be found in the Vivado Synthesis Settings dialog box in the Strategy menu and the strategies are described in this link in the *Vivado Design Suite User Guide: Synthesis* (UG901). When creating the Tcl file for synthesis and implementation, this command is added: `set_property strategy <strategy_name> [get_runs synth_1]`.

Table 8: **System Options** *(cont'd)*

Option	Description
`-trace`	The `-trace` option inserts hardware and software infrastructure into the design to enable tracing functionality.
`-trace-buffer`	The `-trace-buffer` option specifies the trace buffer depth, which must be at least 16 and a power of 2. If this option is not specified, the default value of 1024 is used.
`-trace-no-sw`	The `-trace-no-sw` option inserts hardware trace monitors into the design without instrumenting the software when enabling tracing functionality.
`-vpl-ini <ini_file>`	Specifies an initialization file containing one `-xp <parameter_value>` per line, but do not include the `-xp` option itself. This is equivalent to specify multiple `-xp` options on the command line. Advanced users can use this option to customize the Vivado synthesis and implementation flows.
`-xp <parameter_value>`	Specifies a Vivado synthesis or implementation property or parameter, optionally enclosed in double quotes. The `<parameter_value>` uses one of the following forms to set a Vivado property or parameter, respectively. `"vivado_prop:run.run_name.<prop_name>=<value>"` `"vivado_param:<param_name>=<value>"` Familiarity with the Vivado tool suite is recommended to make the most use of these parameters. The first two examples set a Vivado property to specify a post-synthesis and post-optimization Tcl script, respectively: `vivado_prop:run.synth_1.STEPS.SYNTH_DESIGN.TCL.POST=/path/to/postsynth.tcl"` `"vivado_prop:run.impl_1.STEPS.OPT_DESIGN.TCL.POST=/path/to/postopt.tcl"` The following example sets the maximum number of threads used by Vivado and is equivalent to using the `sds++ -maxthreads` option. It illustrates a method for setting a Vivado parameter: `"vivado_param:general.maxThreads=1"` Advanced users can use the `-xp` option to customize the Vivado synthesis and implementation flows. The `--xp` option is also accepted. Normally, Vivado implementation does not produce a bitstream if there are timing violations. To force `sds++` to skip the timing violation check and continue, allowing you to proceed and correct timing issues later, you can use this parameter: `param:compiler.skipTimingCheckAndFrequencyScaling=1`

Clock ID Values by Platform

For a list of clock ID values by platform, see Appendix A: Clock ID Values by Platform.

Merge Options

Table 9: **Merge Options**

Option	Description
`-merge <input_dir>`	Specify one or more input SDSoC development environment project directories to be merged to create a single design (each contains _sds sub-directory).
`-o <output_dir>`	Specify output directory to create the merged design (if omitted, use the current directory).
`<other_options>`	The following subset of the System Options are supported: • -emulation • -disable-ip-cache • -impl-strategy • -maxjobs • -maxthreads • -mno-boot-files • -mno-bitstream • -remote-ip-cache • -sdcard • -synth-strategy • -vpl-ini • -xp

Compiler Toolchain Support

The SDSoC environment uses the same GNU Arm cross-compiler toolchains included with the Xilinx Software Development Kit (SDK).

The Linaro-based GCC compiler toolchains support the Zynq-7000 SoC and Zynq UltraScale+ MPSoC family devices. This section includes additional information on toolchain usage.

When compiling and linking applications, use only object files and libraries built using the same compiler toolchain and options as those used by the SDSoC environment. All SDSoC provided software libraries and software components (Linux kernel, root filesystem, BSP libraries, and other pre-built libraries) are built with the included toolchains. If you use `sdscc`/`sds++` (referred to as `sds++`) to compile object files, the tools automatically insert a small number of options. If you invoke the underlying toolchains, you must use the same options.

For example, if you use a different Zynq-7000 SoC floating-point application binary interface (ABI), your binary objects are incompatible and cannot be linked with SDSoC Zynq-7000 binary objects and libraries.

The following table summarizes the `sds++` usage of Zynq-7000 SoC toolchains and options. Where options are listed, you need to specify them only if you use the toolchain `gcc` and `g++` commands directly instead of invoking `sds++`.

Table 10: **sds++ Usage with Zynq-7000 SoC**

Usage	Description
Zynq-7000 Arm bare-metal compiler and linker options	`-mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard`
Zynq-7000 Arm bare-metal linker options	`-Wl,--build-id=none -specs=<specfile>` Where the `<specfile>` contains `*startfile:` `crti%O%s crtbegin%O%s`
Zynq-7000 Arm bare-metal compiler	`${SDSOC_install}/SDK/gnu/aarch32/<host>/gcc-arm-none-eabi/bin` Toolchain prefix: `arm-none-eabi` gcc executable: `arm-none-eabi-gcc` g++ executable: `arm-none-eabi-g++`
Zynq-7000 SDSoC bare-metal software (lib, include)	`${SDSOC_install}/aarch32-none`
Zynq-7000 Arm Linux compiler	`${SDSOC_install}/SDK/gnu/aarch32/<host>/gcc-arm-linux-gnueabi/bin` Toolchain prefix: `arm-linux-gnueabihf-` gcc executable: `arm-linux-gnueabihf-gcc` g++ executable: `arm-linux-gnueabihf-g++`
Zynq-7000 SDSoC Linux software (lib, include)	`${SDSOC_install}/aarch32-linux`

The following table summarizes `sds++` usage of Zynq UltraScale+ MPSoC Cortex™-A53 toolchains and options. Where options are listed, you only need to specify them if you use the toolchain `gcc` and `g++` commands directly instead of invoking `sds++`.

Table 11: **sds++ Usage with Zynq UltraScale+ MPSoC Cortex-A53**

Usage	Description
Zynq UltraScale+ MPSoC Arm bare-metal compiler and linker options	`-mcpu=cortex-a53 -DARMR5 -mfloat-abi=hard -mfpu=vfpv3-d16`
Zynq UltraScale+ MPSoC Arm bare-metal linker options	`-Wl,--build-id=none`
Zynq UltraScale+ MPSoC Arm bare-metal compiler	`${SDSOC_install}/SDK/gnu/aarch64/<host>/aarch64-none/bin` Toolchain prefix: `aarch64-none-elf` gcc executable: `aarch64-none-elf-gcc` g++ executable: `aarch64-none-elf-g++`
Zynq UltraScale+ MPSoC SDSoC bare-metal software (lib, include)	`${SDSOC_install}/aarch64-none`

Table 11: **sds++ Usage with Zynq UltraScale+ MPSoC Cortex-A53** *(cont'd)*

Usage	Description
Zynq UltraScale+ MPSoC Arm Linux compiler	`${SDSOC_install}/SDK/gnu/aarch64/<host>/aarch64-linux/bin` Toolchain prefix: `aarch64-linux-gnu-` gcc executable: `aarch64-linux-gnu-gcc` g++ executable: `aarch64-linux-gnu-g++`
Zynq UltraScale+ MPSoC SDSoC Linux software (lib, include)	`${SDSOC_install}/aarch64-linux`

The following table summarizes the `sds++` usage of Zynq UltraScale+ MPSoC Cortex-R5 toolchains and options. Where options are listed, you need to specify them only if you use the toolchain `gcc` and `g++` commands directly instead of invoking `sds++`.

Table 12: **sds++ Usage with Zynq UltraScale+ MPSoC Cortex-R5**

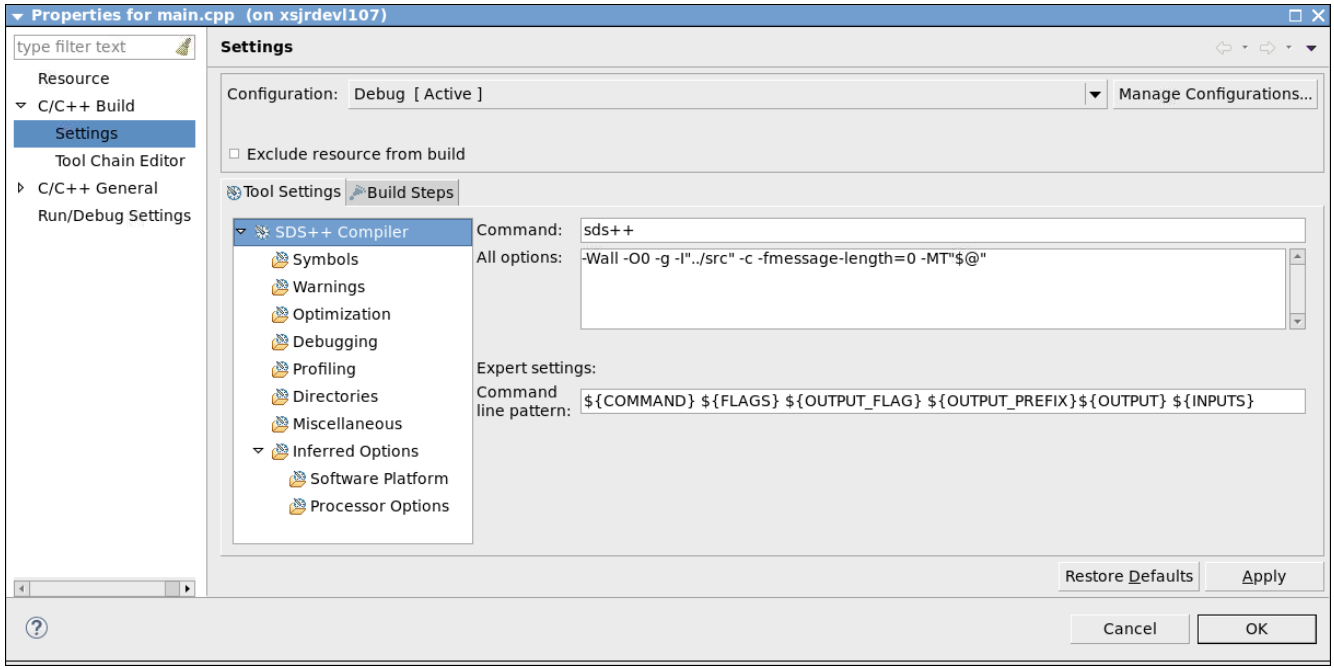
Usage	Description
Zynq UltraScale+ MPSoC Arm bare-metal compiler and linker options	`-mcpu=cortex-r5 -DARMR5 -mfloat-abi=hard -mfpu=vfpv3-d16`
Zynq UltraScale+ MPSoC Arm bare-metal linker options	`-Wl,--build-id=none`
Zynq UltraScale+ MPSoC Arm bare-metal compiler	`${SDSOC_install)/SDK/gnu/armr5/<host>/gcc-arm-none-eabi/bin` Toolchain prefix: `armr5-none-eabi` gcc executable: `armr5-none-eabi-gcc` g++ executable: `armr5-none-eabi-g++`
Zynq UltraScale+ MPSoC SDSoC bare-metal software (lib, include)	`${SDSoC_install}/armr5-none`

IMPORTANT! *When using `sds++` to compile Zynq-7000 source files, be aware that SDSoC tools that are processing and analyzing source files issue errors if they contain NEON instrinsics. If hardware accelerator (or caller) source files contain NEON intrinsics, guard them using the `__SDSCC__` and `__SDSVHLS__` macros.*

For source files that do not contain hardware accelerators or callers but do use NEON intrinsics, you can either compile them directly using the GNU toolchain and link the objects with `sds++`, or you can add the `sds++` command line option `-mno-ir` for these source files. This option prevents clang-based tools from being invoked to create an intermediate representation (IR) used in analysis. You are programmatically aware that they are not required (such as no accelerators or callers).

For the latter solution, if you are using the SDSoC environment, you can apply the option on a per-file basis by right-clicking the source file, select **Properties** and go to the **Settings** dialog box under **C/C++ Build Settings** → **Settings**.

Figure 2: **Build Settings**



emconfigutil (Emulation Configuration) Utility

In the command line flow, it is necessary to create an emulation configuration file and to set certain environment variables prior to running either SW or HW emulation. The emulation configuration file provides the device type and quantity to emulate, and is used by the runtime library during emulation. The emconfigutil utility automates the creation of the emulation file. Use the following steps to setup the emulation configuration file and set the environment variables:

1. Create the emulation configuration file.

Note: When running on real HW, the runtime and drivers query the installed HW to determine the device type and quantity are installed, along with the device characteristics..

The emconfigutil options are given below. The `--platform` option is required. The syntax of the command is:

```
emconfigutil --platform <platform_name> [options]
```

Table 13: **emconfigutil Options**

Option	Valid Values	Description
`--platform`	Target device	Required. Sets target device. For a list of supported devices, refer to *SDAccel Environment Release Notes, Installation, and Licensing Guide* (UG1238).
`--nd`	Any positive integer	Optional. Specifies number of devices. Default is 1.
`--od`	Valid directory	Optional. Specifies output directory. When emulating the design, the `emconfig.json` file must be in the same directory as the host executable.
`--save-temps`	N/A	Optional. Specifies that intermediate files are not deleted and remain after command is executed.
`--xp`	Valid Xilinx parameters and properties	Optional. Specifies additional parameters and properties. For example:`--xp prop:solution.platform_repo_paths=my_dsa_path` Sets the search path for the device specified in `--platform` option.
`-h`	N/A	Prints help messages.

The `emconfigutil` command generates the configuration file `emconfig.json` in the output directory.

When running emulation, the `emconfig.json` file must be in the same directory as the host executable. The following example creates a configuration file targeting two `xilinx_vcu1525_dynamic_5_0` devices.

```
$emconfigutil --platform xilinx_vcu1525_dynamic_5_0
 --nd 2
```

2. Set the `XILINX_SDX` environment variable.

   The `XILINX_SDX` environment needs to be set and must point to the SDAccel installation path for the emulation to work. Below are examples assuming SDAccel™ is installed in `/opt/Xilinx/SDx/2018.3`.

   C Shell:

   ```
   setenv XILINX_SDX /opt/Xilinx/SDx/2018.3
   ```

   Bash:

   ```
   export XILINX_SDX=/opt/Xilinx/SDx/2018.3
   ```

3. Set the emulation mode.

   Setting `XCL_EMULATION_MODE` environment variable to `sw_emu` (software emulation) or `hw_emu` (hardware emulation) changes the application execution to emulation mode. In emulation mode, runtime looks for the file `emconfig.json` in the same directory as the host executable and reads in the target configuration for the emulation runs.

   C Shell:

   ```
   setenv XCL_EMULATION_MODE sw_emu
   ```

   Bash:

   ```
   export  XCL_EMULATION_MODE=sw_emu
   ```

   Not setting the `XCL_EMULATION_MODE` environment variable turns off the emulation mode.

4. Run the emulation.

   With the configuration file `emconfig.json` and `XCL_EMULATION_MODE`, use the following command line to perform emulation:

   ```
   $./host.exe kernel.xclbin
   ```

kernelinfo Utility

Use `kernelinfo` to print the function definitions in the given Xilinx® object file (XO) file (including function arguments and ports).

***Note*:** `kernelinfo` can only be used with the SDAccel™ development environment.

The following command options are available:

- `-h [ --help ]`: Print help message.
- `-x [ --xo_path ]` <arg>: Absolute path to XO file including file name and .xo extension
- `-l [ --log ]` <arg>: By default, information is displayed on the screen. Otherwise, you can use the `--log` option to output the information as a file.
- `-j [ --json ]`: Output the file in JSON format.
- `[input_file]`: XO file. Specify the XO file positionally or use the `--xo_path` option.
- `[output_file]`: output from Xilinx OpenCL™ Compiler. Specify the output file positionally, or use the `--log` option.

platforminfo Utility

Use `platforminfo` to query the platforms that your SDx™ installation is configured to use. The following command options are available to use with `platforminfo`:

Command Options

- `-h [ --help ]`: Print help message and exit.
- `-k [ --keys ]`: Get keys for a given platform. Returns a list of JSON paths.
- `-l [ --list ]`: List platforms. Searches the user repo paths $PLATFORM_REPO_PATHS and then the install locations to find `.xpfm` files.
- `-e [ --extended ]`: List platforms with extended information. Use with '--list'.
- `-d [ --hw ]` <arg>: Hardware platform definition (`*.dsa`) on which to operate. The value must be a full path, including file name and `.dsa` extension.
- `-s [ --sw ]` <arg>: Software platform definition (`*.spfm`) on which to operate. The value must be a full path, including file name and `.spfm` extension.
- `-p [ --platform ]` <arg>: Xilinx® platform definition (`*.xpfm`) on which to operate. The value for `--platform` can be a full path including file name and `.xpfm` extension, as shown in example 1 below. If supplying a file name and `.xpfm` extension without a path, this utility will search only the current working directory. You can also specify just the base name for the platform. When the value is a base name, this utility will search the $PLATFORM_REPO_PATHS, and the install locations, to find a corresponding `.xpfm` file, as shown in example 2 below.

```
Example 1: --platform /opt/xilinx/platforms/xilinx_u200_xdma_201830_1.xpfm

Example 2: --platform xilinx_u200_xdma_201830_1
```

- `-o [ --output ]` <arg>: Specify an output file to write the results to. By default the output is returned to the terminal (stdout).
- `-j [ --json ]` <arg>: Specify JSON format for the generated output. When used with no value, the `platforminfo` utility prints the entire platform in JSON format. This option also accepts an argument that specifies a JSON path, as returned by the `-k` option. The JSON path, when valid, is used to fetch a JSON subtree, list, or value.

```
Example 1:
platforminfo --json="hardwarePlatform" --platform <platform base name>

Example 2: Specify the index when referring to an item in a list.
platforminfo --json="hardwarePlatform.devices[0].name" --platform
```

```
<platform base name>

Example 3: When using the short option form (-j), the value must follow
immediately.
platforminfo -j"hardwarePlatform.systemClocks[]" -p <platform base name>
```

- `-v [ --verbose ]`: Specify more detailed information output. The default behavior is to produce a human-readable report containing the most important characteristics of the specified platform.

***Note*:** Except when using the `--help` or `--list` options, a platform must be specified. You can specify the platform using the `--platform` option, or using either `--hw`, `--sw`. You can also simply insert the platform name or full path into the command line positionally.

Examples

The following example writes JSON formatted output for the platform to the specified file:

```
platforminfo -j xilinx_vcu1525_xdma_201830_1 -o ./vcu1525.json
```

This example returns the metadata keys for the specified platform, and the second command writes the information to the specified file:

```
platforminfo -k zcu106
platforminfo -k zcu102 -o ./zcu106_keys.txt
```

The following returns the System Clocks on the specified platform:

```
platforminfo -j"hardwarePlatform.systemClocks[]" -p
xilinx_u250_xdma_201830_1
```

xbutil (Xilinx Board Utility)

IMPORTANT! *The `xbutil` utility replaces the `xbsak` utility, which is being deprecated.*

The Xilinx® Board Utility (`xbutil`) is a standalone command line utility that is included with the Xilinx Run Time (XRT) installation package. It includes multiple commands to validate and identify the installed card(s) along with additional card details including DDR, PCIe®, shell name (DSA), and system information. This information can be used for both card administration and application debugging. Some of these include:

- Card administration tasks:
  - Flash card configuration memory of the card.
  - Reset hung cards.
  - Query card status, sensors, and PCI Express AER registers.
- Debug operations:
  - Download the SDAccel™ binary (`.xclbin`) to FPGA.
  - Test DMA for PCIe bandwidth.
  - Show status of compute units.

The `xbutil` command line format is:

```
xbutil <command> [options]
```

where the available commands are given below. Specific command options are detailed in the respective command topics:

- clock
- dmatest
- flash
- flash scan
- help
- list
- mem read

- mem write
- program
- query
- reset
- scan
- status
- top
- validate

To run the `xbutil` command without prepending the path `/opt/xilinx/xrt/bin/`, run the following command.

Use the following command in csh shell:

```
$ source /opt/xilinx/xrt/setup.csh
```

Use the following command in bash shell:

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

Note: The sudo access is required for the `flash` option.

clock

The `clock` command allows you to change the clock frequencies driving the computing units. Note that your compute units must be capable or running at the specified clock. You can modify both clock1 and clock2 using this command.

It has the following options:

- `-d <card_id>` (Optional): Specifies the target card. Default = 0 if not specified.
- `-f <clock1_freq>` (Required): Specifies clock frequency (in MHz) for the first clock. All platforms have this clock.
- `-g <clock2_freq>` (Optional): Specifies clock frequency (in MHz) for the second clock. Some platforms may not have this clock.

For example, to change clock1 in card 0 to 100 MHz, run the following command:

```
$ xbutil clock -d 0 -f 100
```

Similarly, to change two clocks in card 0, such that clock1 is set to 200 MHz and clock2 is set to 250 MHz, run this command:

```
$ xbutil clock -d 0 -f 200 -g 250
```

The following example is an output after running this command:

```
INFO: Found total 1 card(s), 1 are usable
INFO: xbutil clock succeeded.
```

dmatest

The `dmatest` command performs throughput data transfer tests between the host machine and global memory on a specified card. Note, it is necessary to download an `xclbin` on the card prior to running `dmatest`, else running this command returns an error. The `dmatest` command only performs throughput tests on those DDR banks accessed by the `xclbin` downloaded to the card.

The command has the following options:

- `-d card_id` (Optional): Specifies the target card. Default = 0 if not specified.
- `-b blocksize` (Optional): Specifies the test block size (in KB). Default = 65536 (KB) if not specified. The block size can be specified in both decimal or hexadecimal formats. For example, both `-b 1024` and `-b 0x400` set the block size to 1024 KB.

To run the `dmatest` command, enter the following:

```
$ xbutil dmatest
```

An example of the command output with an `xclbin` using DDR banks 0, 1, 2, and 3 is shown below:

```
INFO: Found total 1 card(s), 1 are usable
Total DDR size: 65536 MB
Reporting from mem_topology:
Data Validity & DMA Test on bank0
Host -> PCIe -> FPGA write bandwidth = 11341.5 MB/s
Host <- PCIe <- FPGA read bandwidth = 11097.3 MB/s
Data Validity & DMA Test on bank1
Host -> PCIe -> FPGA write bandwidth = 11414.6 MB/s
Host <- PCIe <- FPGA read bandwidth = 10981.7 MB/s
Data Validity & DMA Test on bank2
Host -> PCIe -> FPGA write bandwidth = 11345.1 MB/s
Host <- PCIe <- FPGA read bandwidth = 11189.2 MB/s
Data Validity & DMA Test on bank3
Host -> PCIe -> FPGA write bandwidth = 11121.7 MB/s
Host <- PCIe <- FPGA read bandwidth = 11375.7 MB/s
INFO: xbutil dmatest succeeded.
```

flash

The `flash` command programs the flash configuration memory on the card with a specified deployment shell.

It has the following options:

- `-d <card_id>` (Optional): Specifies the target card ID, else flashes all cards if not specified.
- `-a <shell_name>`: Specifies the name of the deployment shell to program the card or you can set the `shell_name` to `all`. This will attempt to flash all the cards in the system with the installed deployment shell.
- `-t <timestamp>`: Specifies the timestamp associated with the `shell_name`.

For example, to flash the card with a deployment shell called `xilinx_u200_xdma_201820_1` and timestamp `1535712995`, enter the following command:

```
xbutil flash -a xilinx_u200_xdma_201820_1 -t 1535712995
```

Below is an example of the output after the card has been flashed:

```
INFO: ***Found 880 ELA Records
Idcode byte[0] ff
Idcode byte[1] 20
Idcode byte[2] bb
Idcode byte[3] 21
Idcode byte[4] 10
Enabled bitstream guard. Bitstream will not be loaded until flashing is
finished.
Erasing flash..........................................
Programming flash.....................................
Cleared bitstream guard. Bitstream now active.
DSA image flashed succesfully
Cold reboot machine to load the new image on FPGA
```

flash scan

The `flash scan` command returns the current firmware installed on both the card and the system.

It has the following option:

- `-v` (Optional): Verbose output displays additional information about the U250 and U200 cards only.

To run the `flash scan` command, enter the following:

```
xbutil flash scan
```

You should see an output similar to the example below. In this example, the deployment shell name is `xilinx_u200_xdma_201820_1`, the timestamp is `0x000000005b891ee3`, and the BMC version is 1.8. In this output, DSA is referring to the deployment shell, TS is the timestamp, and BMC is referring to the Satellite Controller.

```
Card [0]
    Card BDF:         0000:06:00.1
    Card type:         u200
    Flash type:         SPI
    DSA running on FPGA:
        xilinx_u200_xdma_201820_1,[TS=0x000000005b891ee3],[BMC=1.8]
    DSA package installed in system:
        xilinx_u200_xdma_201820_1,[TS=0x000000005b891ee3],[BMC=1.8]
```

help

The `help` command displays the available `xbutil` commands.

list

The `list` command lists all supported working cards installed on the system along with the card ID. The card ID is used in other `xbutil` commands or in your host code when specifying a particular card.

The output format displays the following three items in this order:

```
[card_id] BDF shell_name
```

There are no options for this command.

To run the `list` command, enter the following:

```
$ xbutil list
```

In this example, the card ID is 0, the BDF is 65:00.0, and the shell name is `xilinx_u250_xdma_201820_1`.

```
INFO: Found total 1 card(s), 1 are usable
[0] 65:00.0 xilinx_u250_xdma_201820_1
```

mem read

The `mem --read` command reads the specified number of bytes starting at a specified memory address and writes the contents into an output file.

- `-a <address>` (Optional): Specifies the starting address (in hexadecimal). Default address is `0x0`.
- `-i <size>`: Specifies the size of memory read (in bytes).
- `-o <file_name>` (Optional): Specifies the output file name. Default output file is `memread.out`.

To run the `mem --read` command to read 256 bytes of data starting at memory address `0x0`, enter the following:

```
$ xbutil mem --read -a 0x0 -i 256 -o read.out
```

This is an example output:

```
INFO: Found total 1 card(s), 1 are usable
INFO: Reading from single bank, 256 bytes from DDR address 0x4000000000
INFO: Read size 0x100 B. Total Read so far 0x100
INFO: Read data saved in file: read.out; Num of bytes: 256 bytes
INFO: xbutil mem succeeded.
```

mem write

The `mem --write` command writes a defined value to a specified memory location and size.

- `-a <address>` (Optional): Specifies the starting address (in hexadecimal). Default address is `0x0`.
- `-i <size>`: Specifies the size of memory read (in bytes).
- `-e <pattern>`: Specifies the pattern (in bytes) to write to memory.

To write the value `0xaa` to 256 locations starting at memory address `0x0`, enter the following:

```
$ xbutil mem --write -a 0x0 -i 256 -e 0xaa
```

This is an example output:

```
INFO: Found total 1 card(s), 1 are usable
INFO: Writing to single bank, 256 bytes from DDR address 0x4000000000
INFO: Writing DDR with 256 bytes of pattern: 0xaa from address 0x4000000000
INFO: xbutil mem succeeded.
```

program

The `program` command downloads an `xclbin` binary to the programmable region on the card.

It has the following options:

- `-d <card_id>` (Optional): Specifies the target card ID. Default = 0 if not specified.
- `-p <xclbin>` (Required): Specifies the `xclbin` binary file to download to the card.

For example, to program the `filter.xclbin` file to card ID one, you would use the following command:

```
$ xbutil program -d 1 -p filter.xclbin
```

This output is displayed after the `xclbin` has been successfully downloaded to the card:

```
INFO: Found total 1 card(s), 1 are usable
INFO: xbutil program succeeded.
```

query

The `query` command returns detailed status information for the specified card.

It has the following option:

- `-d <card_id>` (Optional): Specifies the target card. Default = 0 if not specified.

For example, to query card ID zero, run the following command:

```
xbutil query -d 0
```

An example of the output is given below. The output has been divided into separate sections to better describe the content.

The first section gives details of the installed card including the shell name (DSA name), vendor information, installed DDR, clock, and PCIe information.

```
INFO: Found total 1 card(s), 1 are usable
DSA name
xilinx_u250_xdma_201820_1

Vendor          Device          SubDevice       SubVendor       XMC fw
version
10ee            5004            000e            10ee
2018203

DDR size        DDR count       OCL Frequency   Clock0
```

```
Clock1
64 GB           4                                   300 MHz          500
MHz

PCIe            DMA bi-directional threads          MIG Calibrated
GEN 3x16        2                                   true
```

Card power and thermal information are given next.

```
################################################################################
###
Power
34.9W

PCB TOP FRONT   PCB TOP REAR    PCB BTM FRONT
33 C            28 C            32 C

FPGA Temp       TCRIT Temp      Fan Speed
35 C            33 C            1100 rpm

12V PEX         12V AUX         12V PEX Current 12V AUX Current
11.9V           0.45V           2928mA          32mA

3V3 PEX         3V3 AUX         DDR VPP BOTTOM  DDR VPP TOP
3.36V           3.31V           2.50V           2.50V

SYS 5V5         1V2 TOP         1V8 TOP         0V85
5.49V           1.20V           1.82V           0.85V

MGT 0V9         12V SW          MGT VTT
0.90V           11.9V           1.20V

VCCINT VOL      VCCINT CURR
0.85V           10094mA
```

The firewall provides information when an error has been detected in hardware. This includes a timestamp and the level of the firewall. The firewall has three levels. For more information, see the *SDAccel Environment Debugging Guide* (UG1281). In the below output, there are no detected firewall errors.

```
Firewall Last Error Status:
 Level  0: 0x0 (GOOD)
```

The `xclbin` ID along with the contained Compute Units (CU) are displayed. For each CU, it displays the name, PCIe BAR address, and the status, which can be IDLE, START, and DONE. The output below shows the `xclbin` ID and two CUs both with IDLE status.

```
Xclbin ID:
0x5b996b13

Compute Unit Status:
CU[0]: bandwidth1:kernel_1@0x1800000 (IDLE)
CU[1]: bandwidth2:kernel_2@0x1810000 (IDLE)
```

The memory topology along with the DMA transfer metrics are provided next. The DMA metrics include the transfer of data between the host and card. Host to card transfers are indicated by `h2c`, while card to host transfer are defined by `c2h`.

```
###############################################################################
###
Mem Topology                                        Device Memory
Usage
Tag             Type         Temp         Size     Mem Usage        BO nums
 [0] bank0      MEM_DDR4     31 C         16 GB    0 Byte           0
 [1] bank1      MEM_DDR4     31 C         16 GB    0 Byte           0
 [2] bank2      MEM_DDR4     33 C         16 GB    0 Byte           0
 [3] bank3      MEM_DDR4     31 C         16 GB    0 Byte           0
 [4] PLRAM[0]   **UNUSED**   Not Supp     128 KB   0 Byte           0
 [5] PLRAM[1]   **UNUSED**   Not Supp     128 KB   0 Byte           0
 [6] PLRAM[2]   **UNUSED**   Not Supp     128 KB   0 Byte           0
 [7] PLRAM[3]   **UNUSED**   Not Supp     128 KB   0 Byte           0

Total DMA Transfer Metrics:
  Chan[0].h2c:  49888 MB
  Chan[0].c2h:  22656 MB
  Chan[1].h2c:  8096 MB
  Chan[1].c2h:  22592 MB
```

Finally, here is the successful output:

```
INFO: xbutil query succeeded.
```

reset

The `reset` command resets the programmable region on the card. All running compute units in the region are stopped and reset.

It has the following options:

- `-d <card_id>` (Optional): Specifies the target card ID number. Default = 0 if not specified.
- `-h` (Optional): Performs a hot-reset which resets the card and not just the programmable region. The card is still recognized by the operating system. It is recommended to always use this option.

Enter the following command:

```
$ xbutil reset
```

This output is displayed after the reset has been successfully completed:

```
INFO: Found total 1 card(s), 1 are usable
INFO: xbutil reset succeeded.
```

scan

The `scan` option scans the system, displays drivers, and system information.

It has no options.

To run the `scan` command, enter the following:

```
$ xbutil scan
```

An example of the output is shown below:

```
Linux:4.15.0-33-generic:#36~16.04.1-Ubuntu SMP Wed Aug 15 17:21:05 UTC
2018:x86_64
Distribution: Ubuntu 16.04.5 LTS
GLIBC: 2.23
---
XILINX_OPENCL=""
LD_LIBRARY_PATH="/opt/xilinx/xrt/lib:"
---
[0]mgmt:[65:00.1]:0x5004:0x000e:[xclmgmt:2018.3.2:25857]
[0]user:[65:00.0]:0x5005:0x000e:[xocl_xdma:2018.3.8:128]
```

status

The `status` command displays the status of the debug IPs on the card. Currently, this command can read and report the status of SDx™ performance monitor (SPM) and lightweight AXI protocol checker (LAPC) debug IPs. For more information on adding SPM counters and LAPC in your design, see *SDAccel Environment Debugging Guide* (UG1281).

Below are the available options. If you are running without arguments, it shows the list of available debug IPs.

- `--spm` (Optional): Returns the value of the SPM counters. This option is only applicable if the `xclbin` was compiled with the necessary profiling options.
- `--lapc` (Optional): Returns the values of the violation codes detected by the LAPC. This option is only applicable if `xclbin` was compiled with necessary option to insert AXI protocol checkers at the AXI ports of the compute units.

An example output of the following command is shown below:

```
$ xbutil status
```

```
INFO: Found total 1 card(s), 1 are usable
Number of IPs found: 6
IPs found [<ipname>(<count>)]: spm(2) tracefunnel(1) monitorfifolite(1)
monitorfifofull(1) accelmonitor(1)
Run 'xbutil status' with option --<ipname> to get more information about
the IP
INFO: xbutil status succeeded.
```

An example output using the `--spm` option is shown below:

```
$ xbutil status --spm
```

```
INFO: Found total 1 card(s), 1 are usable
SDx Performance Monitor Counters
CU Name                 AXI Portname  Write Bytes  Write Trans.
interconnect_aximm_host M00_AXI       8192         16
simple_1                M_AXI_GMEM    4096         1024

CU Name                 Read Bytes  Read Trans.  Outstanding Cnt
interconnect_aximm_host 4096        1            0
simple_1                4096        1024         0

CU Name                 Last Wr Addr  Last Wr Data  Last Rd Addr
interconnect_aximm_host 0x0           0             0xe00
simple_1                0x0           0             0xffc

CU Name                 Last Rd Data
interconnect_aximm_host 1483476076
simple_1                1062897
INFO: xbutil status succeeded.
```

When there are no debug IPs in the `xclbin`, you will see a similar output as shown below:

```
INFO: Found total 1 card(s), 1 are usable
INFO: Failed to find any debug IPs on the platform. Ensure that a valid
bitstream with debug IPs (SPM, LAPC) is successfully downloaded.
INFO: xbutil status succeeded.
```

top

The `top` command outputs card statistics including memory topology and DMA transfer metrics. This command is similar to the Linux `top` command. When running, it continues to operate until `q` is entered in the terminal window.

It has the following option:

- `-i <seconds>` (Optional): Refreshes rate (in seconds). Default is 1 second.

To run `top` with a refresh rate of two seconds, enter the following command:

```
$ xbutil top -i 2
```

An output similar to the one below is displayed:

```
Device Memory Usage
[0] bank0     [                    0.00%  ]
[1] bank1     [                    0.00%  ]
[2] bank2     [                    0.00%  ]
[3] bank3     [                    0.00%  ]
[4] PLRAM0    [                    0.00%  ]
[5] PLRAM1    [                    0.00%  ]
[6] PLRAM2    [                    0.00%  ]


Power
25.0W

Mem Topology            Device Memory Usage
Tag             Type         Temp         Size     Mem Usage     Bo nums
[0] bank0        **UNUSED**     32 C         16 GB     0 Byte         0
[1] bank1        MEM_DDR4       37 C         17 GB     0 Byte         0
[2] bank2        **UNUSED**     34 C         18 GB     0 Byte         0
[3] bank3        **UNUSED**     32 C         19 GB     0 Byte         0
[4] PLRAM0       **UNUSED**     Not Supp     128 KB    0 Byte         0
[5] PLRAM1       **UNUSED**     Not Supp     128 KB    0 Byte         0
[6] PLRAM2       **UNUSED**     Not Supp     128 KB    0 Byte         0

Total DMA Transfer Metrics:
Chan[0].h2c:   0  Byte
Chan[0].c2h:   0  Byte
Chan[1].h2c:   0  Byte
Chan[1].c2h:   0  Byte
```

validate

The `validate` command generates a high-level, easy to read summary of the installed card. It validates correct installation by performing the following set of tests:

1. Validates the card found.
2. Checks PCI Express link status.
3. Runs a verify kernel on the card.
4. Performs the following data bandwidth tests:
   a. DMA test: Data transfers between host and FPGA DDR through PCI Express.
   b. DDR test: Data transfers between kernels and FPGA DDR.

It has the following option:

- `-d <card_id>` (Optional): Specifies the target card ID. Default validates all the cards installed in the system.

For example, to run the `validate` command on card ID = 0, enter the following:

```
$ xbutil validate -d 0
```

An example of the returned information is shown below:

```
INFO: Found 1 cards

INFO: Validating card[0]: xilinx_u250_xdma_201820_1
INFO: Checking PCIE link status: PASSED
INFO: Starting verify kernel test:
INFO: verify kernel test PASSED
INFO: Starting DMA test
Host -> PCIe -> FPGA write bandwidth = 11736.3 MB/s
Host <- PCIe <- FPGA read bandwidth = 12190.3 MB/s
INFO: DMA test PASSED
INFO: Starting DDR bandwidth test: ............
Maximum throughput: 45475.441406 MB/s
INFO: DDR bandwidth test PASSED
INFO: Card[0] validated successfully.

INFO: All cards validated successfully.
```

package_xo Command

Description

Kernels written in RTL are compiled using the package_xo command line utility. This utility, similar to xocc -c , generates .xo file which can subsequently used in the xocc linking stage.

Arguments

`-kernel_name <arg>` - (Required) Specifies the name of the RTL kernel.

`-force` - (Optional) Overwrite an existing XO file if one exists.

`-kernel_xml <arg>` - (Optional) Specify the path to an existing kernel XML file.

`-design_xml <arg>` - (Optional) Specify the path to an existing design XML file.

`-ip_directory <arg>` - (Optional) Specify the path to the kernel IP directory.

`-parent_ip_directory` - (Optional) If the kernel IP directory specified contains multiple IPs, specify a directory path to the parent IP where its `component.xml` is located directly below.

`-kernel_files` - (Optional) Kernel file name(s).

`-kernel_xml_args <args>` - (Optional) Generate the `kernel.xml` with the specified function arguments. Each argument value should use the following format:

```
{name:addressQualifier:id:port:size:offset:type:memSize}
```

***Note*:** `memSize` is optional.

`-kernel_xml_pipes <args>` - (Optional) Generate the `kernel.xml` with the specified pipe(s). Each pipe value use the following format:

```
{name:width:depth}
```

`-kernel_xml_connections <args>` - (Optional) Generate the `kernel.xml` file with the specified connections. Each connection value should use the following format:

```
{srcInst:srcPort:dstInst:dstPort}
```

`-xo_path <arg>` - (Required) Specifies the path and file name of the compiled object (XO) container file.

`-quiet` - (Optional) Execute the command quietly, returning no messages from the command. The command also returns TCL_OK regardless of any errors encountered during execution.

***Note*:** Any errors encountered on the command-line, while launching the command, will be returned. Only errors occurring inside the command will be trapped.

`-verbose` - (Optional) Temporarily override any message limits and return all messages from this command.

***Note*:** Message limits can be defined with the set_msg_config command.

Examples

The following example creates the specified XO file containing an RTL kernel of the specified name:

```
package_xo -xo_path /temp/data/rtl_kernel/Vadd_A_B.xo -kernel_name Vadd_A_B
```

sdx_pack Utility

The SDSoC™ tools include the `sdx_pack` command line utility for creating C-Callable IP libraries for linking RTL IP into SDSoC applications using the `sds++` system compiler.

Usage

```
sdx_pack -header <header.h/pp> -ip <component.xml> [-param <name>="value"]
[configuration options]
```

Configuration Options

Table 14: **Configuration Options**

Option	Valid Values	Description
`-header <header.h/.hpp>`	Header file with function declarations, Only one top header file allowed	Required. Header file with function prototype.
`-ip <component.xml>`	N/A	Required. IP packed by the Vivado® IP integrator.
`-control <protocol>[=<port>[:offset]]`	N/A	Required. IP control protocol options: • `AP_CTRL` • `AXI` • `none`
`-func <function_name> -map`	N/A	Specify a list of C-Callable functions associated with the IP instance. For each function, use the `[-func ... -map ... -func-end]` option. To map each software function argument to an IP port, use `-map`.
`-map <sw_name>=<hw_name>:direction[:<offset>[<aximm_name>:<direction>]]`	N/A	Required for using `-func <function_name>`. A software function argument to IP port mapping:`<SW_Name>=<HW_Name>:direction[:offset[<aximm_name>:direction]` For example,
`-primary-clk <clk_interface>=min_clk_period`	N/A	Specify the primary clock interface and its minimum clock period in nanoseconds.
`-derived-clk <clk_interface>=multiplier:divisor`	N/A	Specify the phase-aligned derived clock interface, and its multiplier and divisor in integer.

Table 14: **Configuration Options** *(cont'd)*

Option	Valid Values	Description
`-primary-clk <clk_interface>:min_clk_period`	N/A	Specify the primary clock interface and its minimum clock period in nanoseconds.
`-derived-clk <clk_interface>:multiplier:divisor`	N/A	Specify the phase-aligned derived clock interface, and its multiplier and divisor in integer.
`-target-cpu <cpu_type>`	N/A	Specifies target CPU: • `cortex-a9` • `cortex-a53` • `cortex-r5` • `microblaze`
`-target-family <board_family>`	N/A	Specifies target board family; for example, `zynq` or `zynquplus`.
`-target-os <name>`	N/A	Specifies target Operating System: • `linux` (default) • `standalone` (bare-metal)
`-verbose`	N/A	Prints verbose output to STDOUT.
`-version`	N/A	Prints the `sdx_pack` version information to STDOUT.
`--help`	N/A	Displays information about this command: `sdx_pack --help`
`-query-target <type>`	• family • cpu • os	Query supported `board_family`, `cpu_type`, and `os_type` of the IP.
`-query-interface <interface type>`	• all • aximm • axilite • axis • clock • control • param • misc	Query interfaces and parameters of the IP, support multiple queries.

Table 14: **Configuration Options** *(cont'd)*

Option	Valid Values	Description
`-primary-clk <clk_interface>=min_clk_period`	N/A	Specify the primary clock interface and its minimum clock period in nanoseconds. Only one top primary clock is allowed, `min_clk_period` should be between 0.5 and 1000.0 (i.e., 1 MHz to 2 GHz)
`-derived-clk <clk_interface>=multiplier:divisor`	N/A	Specify the phase-aligned derived clock interface, and its multiplier and divisor in integer.

sdx_pack Example

```
sdx_pack -header count.hpp -ip ../ip/component.xml -func count \
-control AXI=S_AXI:0  -map start_value=S_AXI:in:0x8 -map return=S_AXI:out:4
-func-end \
-target-cpu cortex-a9 -target-os standalone -target-family zynq
```

Where the flags that are used above are defined in the table.

The `sdx_pack` utility automatically generates:

- `<function_name>.o`: Compiled object code for the specified function. This file is generated under the `.Xil/sdx` folder.
- `<function_name>.fcnmap.xml`: Mapping IP ports to function arguments. This file is generated under the `.Xil/rtl` folder.
- `<function_name>.params.xml`: IP parameters. This file is generated under the `.Xil/rtl` folder.
- `<function_name>.cpp`: C++ file with entry point. This file is generated under the `.Xil/rtl` folder.

-query Option usage

```
sdx_pack -query-target <type> -ip component.xml [-target-family
<board_family>] [-target-cpu <cpu_type>]
```

This query only supports meaningful queries instead of all possible combinations. Only one `-query-target <type>` option supported at each query. Specifically, below is what `-query-target <type>` returns. This is consistent with the SDx™ GUI for the application project.

- **`-query-target family`**:

  It ignores all other options and returns all supported board families of the IP. In 2018.3, this family set must be a subset of {artix7, kintex7, kintexu, kintexuplus, spartan7, virtex7, virtexu, virtexuplus, virtexuplusHBM, zynq, zynquplus}.

***Note*:** For a family with an "a" (automotive), "q" (space) prefix, or "l" (low power) suffix, then specify the family without the prefix or suffix. For example, target `azynq`, select `zynq`.

- **`-query-target cpu`:** It ignores all other options except `-target-family <board_family>`.
  - If `-target-family <board_family>` is specified, it returns all CPU types supported under this specific family. In 2018.3, this must be a subset of {cortex-a9, cortex-a53, cortex-r5, microblaze}.
  - If `-target-family <board_family>` is not selected, then it returns all CPU types supported under all supported families. In 2018.3, this must be a subset of {cortex-a9, cortex-a53, cortex-r5, microblaze} {cortex-a9, cortex-a53, cortex-r5, microblaze}.
- **`-query-target os`:**
  - If `-target-cpu <cpu_type>` is specified, it returns all OS types supported under this specific CPU. In 2018.3, this must be a subset of {linux, standalone}.

    If `-target-family <board_family>` is also specified, cpu_type must be a valid CPU in this specific board_family.
  - Else, if `-target-family <board_family>` is specified, it returns all OS types supported under this specific family. In 2018.3, this must be a subset of {linux, standalone}.
  - Else, it returns all OS types supported under all supported families. In 2018.3, this must be a subset of {linux, standalone}

-map Option Usage

In this example, the hardware name for the AXI4-Lite interface is `s_axi_AXILiteS`.

Scalars are always mapped onto AXI4-Lite interfaces:

- To map an input scalar (e.g., int a), use `-map a=s_axi_AXILiteS:in:offset`.
- To map an output scalar (e.g., int *a, or int &a), use `-map a=s_axi_AXILiteS:out:offset`.
- To map a return scalar (the return type can only be scalar), use `-map return=s_axi_AXILiteS:out:offset`.

Arrays (e.g., int a[N]) can be mapped onto any type of interface. The exact interface and direction is specified by the IP.

- To map an array onto AXI4, use `-map a=s_axi_AXILiteS:in:offset,<a_hwName>:direction`. Do not map onto AXI4 (m_axi) when `control none` is used.
- To map an array onto AXI4-Stream, use `-map a=<a_hwName>:direction`.

- To map an array (a small, constant-size array) onto AXI4-Lite, use `-map a=s_axi_AXILiteS:in:offset`.

***Note*:** The array must be a one-dimensional array with a constant size. For example, int a[N], where N must be a constant (which can be either #define N 16 or constant int N = 16).

xclbinutil Utility

This utility operates on a xclbin produced by Xilinx® OpenCL™ Compiler. See the SDAccel Environment User Guide (UG1023) for more information.

For example:

- Reporting xclbin information: `xclbinutil --info --input binary_container_1.xclbin`
- Extracting the bitstream image: `xclbinutil --dump-section BITSTREAM:RAW:bitstream.bit --input binary_container_1.xclbin`
- Extracting the build metadata: `xclbinutil --dump-section BUILD_METADATA:HTML:buildMetadata.json --input binary_container_1.xclbin`
- Removing a section: `xclbinutil --remove-section BITSTREAM --input binary_container_1.xclbin --output binary_container_modified.xclbin`
- `xclbinutil --input verify.xclbin --dump-section IP_LAYOUT:JSON:ip_layout_orig.json`
- `xclbinutil --input verify.xclbin --output verifyout.xclbin --append-section IP_LAYOUT:JSON:ip_layout_append.json`
- `xclbinutil --input verifyout.xclbin --dump-section IP_LAYOUT:JSON:ip_layout_modified.json`
- `xclbinutil --input verify.xclbin --info --verbose`

Command Options

- `-h [ --help ]`: Print help messages
- `i [ --input ]` <arg>: Input file name. Reads xclbin into memory.
- `-o [ --output ]` <arg>: Output file name. Writes in memory xclbin image to a file.
- `-v [ --verbose ]`: Display verbose/debug information.
- `-q [ --quiet ]`: Minimize reporting information.
- `--migrate-forward`: Migrate the xclbin archive forward to the new binary format.
- `--remove-section`: <arg> Section name to remove.
- `--add-section` <arg>: Section name to add. Format: `<section>:<format>:<file>`

- `--dump-section` <arg> Section to dump. Format: `<section>:<format>:<file>`
- `--replace-section` <arg> Section to replace.
- `--key-value` <arg> Key value pairs. Format: [USER|SYS]:<key>:<value>
- `--remove-key` <arg> Removes the given user key from the xclbin archive.
- `--add-signature` <arg> Adds a user defined signature to the given xclbin image.
- `--remove-signature` Removes the signature from the xclbin image.
- `--get-signature` Returns the user defined signature (if set) of the xclbin image.
- `--info` Report accelerator binary content. Including: generation and packaging data, kernel signatures, connectivity, clocks, sections, etc.
- `--list-names` List all possible section names (Stand Alone Option)
- `--version` Version of this executable.
- `--force` Forces a file overwrite.

<section>:<format>:<file> Syntax Information

- **<section>:** The section to add or dump (e.g., `BUILD_METDATA`, `BITSTREAM`, etc.)

***Note*:** If a JSON format is being used, this value can be empty. If so, then the JSON *metadata* will determine the section it is associated with. In addition, only sections that are found in the JSON file will be reported.

- **<format>:** The format to be used. Currently, there are three formats available:
  - **RAW:** Binary Image
  - **JSON:** JSON file format
  - **HTML:** Browser visible
- **<file>:** The name of the input/output file to use.

For example, `--add-section BITSTREAM:RAW:mybitstream.bit`.

Clock ID Values by Platform

Platform	Value of <n>
zc702	0 – 166 MHz
	1 – 142 MHz
	2 – 100 MHz
	3 – 200 MHz
zc706	0 – 166 MHz
	1 – 142 MHz
	2 – 100 MHz
	3 – 200 MHz
zed	0 – 166 MHz
	1 – 142 MHz
	2 – 100 MHz
	3 – 200 MHz
zcu102	0 – 75 MHz
	1 – 100 MHz
	2 – 150MHz
	3 – 200 MHz
	4 – 300 MHz
	5 – 400 MHz
	6 – 600 MHz
zcu104	0 – 75 MHz
	1 – 100 MHz
	2 – 150MHz
	3 – 200 MHz
	4 – 300 MHz
	5 – 400 MHz
	6 – 600 MHz

Platform	Value of <n>
zcu106	0 – 75 MHz
	1 – 100 MHz
	2 – 150MHz
	3 – 200 MHz
	4 – 300 MHz
	5 – 400 MHz
	6 – 600 MHz

Appendix B

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

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

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

References

These documents provide supplemental material useful with this guide:

SDAccel Documents

1. *SDAccel Environment User Guide* (UG1023)
2. *SDAccel Environment Profiling and Optimization Guide* (UG1207)
3. *SDAccel Environment Getting Started Tutorial* (UG1021)
4. *SDAccel Environment Debugging Guide* (UG1281)

SDSoC Documents

1. *SDSoC Environment User Guide* (UG1027)
2. *SDSoC Environment Profiling and Optimization Guide* (UG1235)
3. *SDSoC Environment Tutorial: Introduction* (UG1028)
4. *SDSoC Environment Platform Development Guide* (UG1146)

Additional Documents

1. *SDx Pragma Reference Guide* (UG1253)
2. *Xilinx OpenCV User Guide* (UG1233)
3. *Platform Cable USB II Data Sheet* (DS593)

More Resources

1. Xilinx® licensing website: https://www.xilinx.com/getproduct
2. SDSoC Developer Zone: https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html.
3. SDAccel Developer Zone: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html
4. *Xilinx End-User License Agreement* (UG763)
5. *Third Party End-User License Agreement* (UG1254)

Training Resources

1. SDSoC Development Environment and Methodology
2. Advanced SDSoC Development Environment and Methodology

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

Copyright