

LIBXSTREAM

LIBXSTREAM is an OpenCL-based accelerator backend library providing streams, events, and device memory management for GPU offloading. It implements the DBCSR ACC interface, making it a drop-in OpenCL backend for CP2K/DBCSR.

Targets OpenCL devices across vendors (Intel, AMD, NVIDIA). Depends on LIBXS for utility infrastructure.

Build

LIBXS must be a sibling directory (controlled by LIBXSROOT). An OpenCL SDK must be available.

Library -- build both LIBXS and LIBXSTREAM as separate libraries:

```
git clone https://github.com/hfp/libxs.git
git clone https://github.com/hfp/libxstream.git
```

```
cd libxs && make GNU=1 -j $(nproc)
cd ../libxstream && make GNU=1 -j $(nproc)
```

This produces `lib/libxstream.a` and `lib/libxstream.so`.

Header-only (explicit) -- include `libxstream_source.h` (no separate library needed for either LIBXSTREAM or LIBXS). Safe to include from multiple translation units:

```
#include <libxstream/libxstream_source.h>
```

Header-only (implicit) -- compile with `-DLIBXSTREAM_SOURCE`. Any LIBXSTREAM or LIBXS public header then automatically includes the implementation. No special include order is required. `-DLIBXSTREAM_SOURCE` implies `-DLIBXS_SOURCE`; LIBXS can also be made header-only independently with `-DLIBXS_SOURCE`.

The library is compiled for SSE4.2 by default but dynamically dispatches to the best ISA available at runtime (up to AVX-512). Use `SSE=0` to compile natively for the build host.

Variable	Default	Description
GNU	0	Use GNU GCC-compatible compiler
DBG	0	Debug build
SYM	0	Include debug symbols (-g)
SSE	1	x86 baseline: 0=native, 1=SSE4.2 (portable)

pkg-config support: `lib/libxstream.pc`.

Installation

Install into a chosen prefix (LIBXS must be built first):

```
make GNU=1 -j $(nproc) install PREFIX=$HOME/libxstream
```

This installs headers, the static and shared libraries, and the header-only source tree under `PREFIX`.

Out-of-tree builds are also supported:

```
mkdir /tmp/libxstream-build && cd /tmp/libxstream-build
make -j $(nproc) -f /path/to/libxstream/Makefile
```

API

The public C API is declared in `libxstream/libxstream.h`. All implementation details are sealed behind opaque types.

Initialization

```
int libxstream_init(void);
int libxstream_init_config(const libxstream_init_config_t* cfg);
void libxstream_init_config_default(libxstream_init_config_t* cfg);
int libxstream_finalize(void);
```

`libxstream_init()` initializes with environment/defaults. `libxstream_init_config()` allows explicit control over USM level, device selection, and verbosity before the runtime starts:

```
libxstream_init_config_t cfg;
libxstream_init_config_default(&cfg); /* -1 = use env/default */
cfg.usm = 1; /* 0:off, 1:Intel USM, 2:SVM coarse, 3:SVM caps */
cfg.device = 0; /* device index (-1: env/first) */
cfg.verbosity = 2;
libxstream_init_config(&cfg);
```

Passing NULL to `libxstream_init_config` is equivalent to calling `libxstream_init`.

Devices

```
int libxstream_device_count(int* ndevices);
int libxstream_device_set_active(int device_id);
int libxstream_device_sync(void);
```

Streams

```
int libxstream_stream_create(libxstream_stream_t** stream_p,
                             const char* name, int priority);
int libxstream_stream_destroy(libxstream_stream_t* stream);
int libxstream_stream_sync(libxstream_stream_t* stream);
int libxstream_stream_wait_event(libxstream_stream_t* stream,
                                 libxstream_event_t* event);
```

Events

```
int libxstream_event_create(libxstream_event_t** event_p);
int libxstream_event_destroy(libxstream_event_t* event);
int libxstream_event_record(libxstream_event_t* event,
                            libxstream_stream_t* stream);
int libxstream_event_query(libxstream_event_t* event,
                           libxstream_bool_t* has_occurred);
int libxstream_event_sync(libxstream_event_t* event);
```

Memory

Device and host memory allocation, transfers (H2D, D2H, D2D), and initialization:

```
int libxstream_mem_allocate(void** dev_mem, size_t nbytes);
int libxstream_mem_deallocate(void* dev_mem);
int libxstream_mem_host_allocate(void** host_mem, size_t nbytes,
                                 libxstream_stream_t* stream);
int libxstream_mem_host_deallocate(void* host_mem,
                                   libxstream_stream_t* stream);
int libxstream_mem_copy_h2d(const void* host_mem, void* dev_mem,
                            size_t nbytes, libxstream_stream_t* stream);
int libxstream_mem_copy_d2h(const void* dev_mem, void* host_mem,
                            size_t nbytes, libxstream_stream_t* stream);
int libxstream_mem_copy_d2d(const void* src, void* dst,
                            size_t nbytes, libxstream_stream_t* stream);
int libxstream_mem_zero(void* dev_mem, size_t offset, size_t nbytes,
                        libxstream_stream_t* stream);
```

DBCSR Compatibility

The header `libxstream/libxstream_dbcsr.h` provides the `c_dbcsr_acc_*` symbols expected by DBCSR, allowing LIBXSTREAM to serve as a drop-in accelerator backend.

CP2K Offload Interface

The header `libxstream/libxstream_cp2k.h` implements CP2K's offload runtime interface -- the `offload*` functions for general GPU operations (memory, streams, events, synchronization).

Samples

Each sample has its own Makefile under `samples/`:

```
cd samples/smm && make GNU=1
cd samples/ozaki && make GNU=1
```

SMM -- Small Matrix Multiplication

Implements the ACC LIBSMM interface for batched small matrix multiply and transpose on OpenCL devices. Includes an auto-tuning framework and pre-tuned parameter sets for A100, BMG, GH200, H100, Mi250, P100, PVC, and V100. See `samples/smm/README.md`.

Ozaki -- High-Precision GEMM

Ozaki scheme for high-precision GEMM emulation, fully offloaded to OpenCL. Two schemes (mantissa slicing and CRT) with automatic detection of Intel XMV matrix engines. See `samples/ozaki/README.md`. The CPU-side GEMM wrapper is part of LIBXS Ozaki.

Stencil -- BF16-DPAS Finite Difference

Seismic wave propagation (RTM/FWI) via dimension-split stencils computed as batched small GEMMs on BF16 DPAS/XMV tensor units. Achieves FP32 accuracy through Dekker splitting. Supports isotropic and TTI operators, multiple factorization methods (sparse, dense cascade, hybrid), and runtime kernel specialization via a thread-safe registry. Includes a scalar proxy for non-DPAS hardware. See `samples/stencil/README.md`.

License

BSD 3-Clause

LIBXSTREAM Domains

CP2K Offload Interface

`libxstream/libxstream_cp2k.h` implements CP2K's offload runtime interface — the hardware-abstraction layer that CP2K uses for GPU-accelerated operations beyond DBCSR (grid integration, PW operations, etc.). The original interface provides `static inline` wrappers for CUDA, HIP, and OpenCL; LIBXSTREAM replaces the OpenCL path with a dedicated translation unit (`src/libxstream_cp2k.c`) that routes directly through LIBXSTREAM's internal API.

Relationship to the DBCSR Interface

CP2K's code has two accelerator interfaces:

Interface	Header	Purpose		DBCSR ACC	libxstream_dbcsr.h	Sparse matrix operations (DBCSR library)
Offload Runtime	libxstream_cp2k.h	General offload (memory, streams, events, synchronization)				

The DBCSR adapter (`src/libxstream_dbcsr.c`) uses opaque `void*` handles and translates to LIBXSTREAM's typed API. The offload runtime adapter does the same but uses CP2K's `offloadStream_t/offloadEvent_t` typedefs (also `void*`). Both share the underlying LIBXSTREAM implementation.

API

The header is self-contained (C99, no LIBXSTREAM headers required) and provides opaque handle types, an error-checking macro, and functions covering five domains.

```
typedef void* offloadStream_t;
typedef void* offloadEvent_t;
typedef int   offloadError_t;

#define offloadSuccess EXIT_SUCCESS
```

```
const char* offloadGetErrorName(offloadError_t error);
offloadError_t offloadGetLastError(void);
```

`offloadGetErrorName` maps error codes to OpenCL error strings via `libxstream_opencl_strerror`. `offloadGetLastError` consumes and clears the last recorded error.

The `OFFLOAD_CHECK` macro aborts on failure after printing the error name and source location:

```
OFFLOAD_CHECK(offloadMalloc(&ptr, nbytes));
```

```
void offloadStreamCreate(offloadStream_t* stream);
void offloadStreamDestroy(offloadStream_t stream);
void offloadStreamSynchronize(offloadStream_t stream);
void offloadStreamWaitEvent(offloadStream_t stream, offloadEvent_t event);
```

`offloadStreamCreate` creates a stream with default priority (`LIBXSTREAM_STREAM_DEFAULT`).

```
void offloadEventCreate(offloadEvent_t* event);
void offloadEventDestroy(offloadEvent_t event);
void offloadEventRecord(offloadEvent_t event, offloadStream_t stream);
void offloadEventSynchronize(offloadEvent_t event);
bool offloadEventQuery(offloadEvent_t event);
```

```
void offloadMalloc(void** ptr, size_t size);
void offloadFree(void* ptr);
void offloadMallocHost(void** ptr, size_t size);
void offloadFreeHost(void* ptr);
```

```
void offloadMemcpyAsyncHtoD(void* ptr_dev, const void* ptr_hst, size_t size, offloadStream_t stream);
void offloadMemcpyAsyncDtoH(void* ptr_hst, const void* ptr_dev, size_t size, offloadStream_t stream);
void offloadMemcpyAsyncDtoD(void* dst, const void* src, size_t size, offloadStream_t stream);
void offloadMemcpyHtoD(void* ptr_dev, const void* ptr_hst, size_t size);
void offloadMemcpyDtoH(void* ptr_hst, const void* ptr_dev, size_t size);
void offloadMemsetAsync(void* ptr, int val, size_t size, offloadStream_t stream);
void offloadMemset(void* ptr, int val, size_t size);
```

The synchronous variants (`offloadMemcpyHtoD`, `offloadMemcpyDtoH`, `offloadMemset`) pass a `NULL` stream. `offloadMemsetAsync` supports arbitrary fill values via `libxstream_opencl_memset`.

```
void offloadDeviceSynchronize(void);
```

```
void offloadMemcpyToSymbol(const void* symbol, const void* src, size_t count);
void offloadEnsureMallocHeapSize(size_t required_size);
```

These are CUDA-specific operations (constant-memory writes and device-heap sizing) that have no direct OpenCL equivalent. They are currently stubs guarded by assertions. CP2K's OpenCL path disables the GPU grid subsystem (`__NO_OFFLOAD_GRID`) that would call them.

See Also

- LIBXSTREAM API (`libxstream/libxstream.h`) — the underlying OpenCL backend API
- DBCSR ACC Interface — the DBCSR adapter layer
- CP2K `offload_runtime.h` — upstream interface definition

DBCSR ACC Interface

`libxstream/libxstream_dbcsr.h` implements the DBCSR ACC interface — the accelerator backend contract defined by the DBCSR sparse-matrix library used in CP2K. By providing this interface on top of LIBXSTREAM's OpenCL runtime, the library acts as a **drop-in OpenCL backend** for DBCSR: no changes to DBCSR or CP2K source code are required.

Purpose

DBCSR expects every accelerator backend to expose a flat C API with the `c_dbcsr_acc_*` prefix covering five domains:

Domain	Functions	Description		Initialization	<code>c_dbcsr_acc_init</code> , <code>c_dbcsr_acc_finalize</code>	Library setup and teardown
	Devices	<code>c_dbcsr_acc_get_ndevices</code> , <code>c_dbcsr_acc_set_active_device</code> , <code>c_dbcsr_acc_device_synchronize</code>				Device enumeration, selection, and synchronization
	Streams	<code>c_dbcsr_acc_stream_create</code> , <code>c_dbcsr_acc_stream_destroy</code> , <code>c_dbcsr_acc_stream_sync</code> , <code>c_dbcsr_acc_stream_wait_event</code> , <code>c_dbcsr_acc_stream_priority_range</code>				Asynchronous command queues
	Events	<code>c_dbcsr_acc_event_create</code> , <code>c_dbcsr_acc_event_destroy</code> , <code>c_dbcsr_acc_event_record</code> , <code>c_dbcsr_acc_event_query</code> , <code>c_dbcsr_acc_event_synchronize</code>				Fine-grained synchronization primitives
	Memory	<code>c_dbcsr_acc_dev_mem_allocate</code> , <code>c_dbcsr_acc_dev_mem_deallocate</code> , <code>c_dbcsr_acc_dev_mem_set_ptr</code> , <code>c_dbcsr_acc_host_mem_allocate</code> , <code>c_dbcsr_acc_host_mem_deallocate</code> , <code>c_dbcsr_acc_memcpy_h2d</code> , <code>c_dbcsr_acc_memcpy_d2h</code> , <code>c_dbcsr_acc_memcpy_d2d</code> , <code>c_dbcsr_acc_memset_zero</code> , <code>c_dbcsr_acc_dev_mem_info</code>				Device/host allocation, transfers (H2D, D2H, D2D), and memory queries

Every function delegates to the corresponding `libxstream_*` routine (e.g., `c_dbcsr_acc_stream_create` calls `libxstream_stream_create`), translating between DBCSR's opaque `void*` handles and LIBXSTREAM's typed `libxstream_stream_t` / `libxstream_event_t` pointers.

Profiling

The header also declares `c_dbcsr_timeset` and `c_dbcsr_timestop`, which are DBCSR's Fortran-side timer callbacks. When profiling is enabled (`LIBXSTREAM_PROFILE_DBCSR`), every ACC function is bracketed by these calls so that individual backend operations appear in DBCSR's timing report.

Utility Macros

Macro	Description		<code>DBC_SR_STRINGIFY(SYMBOL)</code>	Stringifies a preprocessor token		<code>DBC_SR_CONCATENATE(A, B)</code>	
Concatenates two preprocessor tokens							
			<code>DBC_SR_MARK_USED(x)</code>	Silences unused-variable warnings			

See Also

- LIBXSTREAM API (`libxstream/libxstream.h`) — the underlying OpenCL backend API
- DBCSR ACC specification — upstream interface definition

Visit LIBXS

OpenCL Backend

`libxstream/libxstream_opencl.h` is the internal OpenCL layer that powers every public `libxstream_*` function. It owns the OpenCL platform/device/context lifecycle, memory management, kernel compilation, and error handling. Sample code and other LIBXSTREAM extensions (e.g., LIBSMM, Ozaki) include this header to access the OpenCL runtime directly.

Compile-Time Configuration

The header is guarded by `__OPENCL` (set automatically when `__OFFLOAD_OPENCL` is defined). Key compile-time knobs:

Macro	Default	Description		<code>LIBXSTREAM_MAXALIGN</code>	2 MB	Maximum alignment for device allocations	
<code>LIBXSTREAM_BUFFERSIZE</code>							
8 KB							
Internal scratch-buffer size							
				<code>LIBXSTREAM_MAXSTRLEN</code>	48	Maximum string length for names	
<code>LIBXSTREAM_MAXNDEVS</code>							
64							
Maximum number of OpenCL devices							
				<code>LIBXSTREAM_MAXNITEMS</code>	1024	Per-thread maximum item count	
<code>LIBXSTREAM_USM</code>							
SVM coarse-grain							
Runtime Unified Shared Memory level (unset = OpenCL 2.0 SVM coarse-grain with non-USM fallback, 0 = off, 1 = Intel USM, 2 = OpenCL 2.0 SVM coarse-grain, 3 = OpenCL 2.0 SVM reported caps)							

Data Types

libxstream_opencl_config_t The central singleton (`libxstream_opencl_config`) populated by `libxstream_init`. It holds:

- **Device table** — ordered array of discovered `cl_device_id` entries.
- **Active device** (`libxstream_opencl_device_t`) — context, default stream, error slot, OpenCL standard level, workgroup limits, memory caps, vendor flags, and optional USM function pointers.
- **Resource pools** — lock objects, streams, events, memory-pointer registrations, and a host-memory pool (`libxs_malloc_pool_t`).
- **Runtime switches** — verbosity, async mode, debug/dump level, profiling, execution hints, and workaround level.
- **Histograms** — optional transfer-time histograms for H2D, D2H, and D2D copies.

libxstream_opencl_stream_t / libxstream_event_t Thin wrappers around `cl_command_queue` and `cl_event` respectively. Streams additionally carry a thread-ID and optional priority.

libxstream_opencl_info_memptr_t Associates a `cl_mem` buffer object with its host-side pointer, used to translate between SVM/USM pointers and buffer-based memory.

libxstream_opencl_atomic_fp_t Enumerates floating-point atomics support: none, 32-bit, or 64-bit.

Error Handling Macros

Macro	Description		<code>CL_CHECK(RESULT, CALL)</code>	Execute an OpenCL call; on failure record the error code and human-readable name
	<code>CL_ERROR_REPORT(NAME)</code>			Print the last error to stderr (if verbosity is enabled)
	<code>CL_RETURN(RESULT, NAME)</code>			Return from function, reporting the error if non-zero

Key Functions

Device and Context	Function	Description		<code>libxstream_opencl_set_active_device</code>	Internal device activation (lock-aware)
		<code>libxstream_opencl_create_context</code>			Create an OpenCL context for a given device
		<code>libxstream_opencl_device_name</code>			Return device name, platform name, and UID
		<code>libxstream_opencl_device_level</code>			Query OpenCL version and device type
		<code>libxstream_opencl_device_vendor</code>			Confirm a device's vendor string
		<code>libxstream_opencl_device_ext</code>			Check for required OpenCL extensions
		<code>libxstream_opencl_device_uid</code>			Capture or compute a unique device identifier
		<code>libxstream_opencl_info_devmem</code>			Query free/total/local device memory

Memory	Function	Description		<code>libxstream_opencl_info_devpnr</code>	Look up a device-pointer registration (read-only)
		<code>libxstream_opencl_info_devpnr_modify</code>			Look up a device-pointer registration (writable)
		<code>libxstream_opencl_info_hostpnr</code>			Look up a host-pointer registration
		<code>libxstream_opencl_memset</code>			Fill device memory with an arbitrary byte pattern
		<code>libxstream_opencl_use_cmern</code>			Whether OpenCL constant-memory hints apply
		<code>libxstream_opencl_set_kernel_ptr</code>			Set a pointer kernel argument (USM-aware)

Kernel Build	Function	Description		<code>libxstream_opencl_program</code>	Compile an OpenCL program from source, file, or binary
		<code>libxstream_opencl_kernel_query</code>			Extract a named kernel from a compiled program
		<code>libxstream_opencl_kernel</code>			Convenience: build + extract + release in one call
		<code>libxstream_opencl_kernel_flags</code>			Assemble combined build flags from params, options, and extras
		<code>libxstream_opencl_defines</code>			Merge user defines with internal definitions
		<code>libxstream_opencl_flags_atomics</code>			Generate compiler flags for FP-atomic extensions

Streams, Events, and Timing	Function	Description		<code>libxstream_opencl_stream</code>	Find an existing stream for a thread-ID
		<code>libxstream_opencl_stream_default</code>			Return the device's default (internal) stream
		<code>libxstream_opencl_device_synchronize</code>			Per-thread device synchronization
		<code>libxstream_opencl_duration</code>			Measure elapsed seconds from a <code>cl_event</code>

Error Utilities		Function		Description				<code>libxstream_openccl_strerror</code>		Map a <code>cl_int</code> error code to a string		
<code>libxstream_openccl_error_consume</code>		Clear and return the last recorded error										

See Also

- LIBXSTREAM API (`libxstream/libxstream.h`) — public API built on top of this layer
- DBCSR ACC Interface — the DBCSR compatibility shim

Start Presentation

Appendix

Scripts

Helper scripts for building, inspecting, and maintaining LIBXSTREAM.

`tool_checkabi.sh`

Checks the ABI (Application Binary Interface) stability of the LIBXSTREAM shared library. The script uses `nm` to extract exported symbols from `lib/*.so` (or `lib/*.a` as fallback) and verifies that no previously published symbol has been removed or renamed compared to an earlier baseline (`.abi.txt`). Non-conforming symbol names cause an immediate error.

`scripts/tool_checkabi.sh`

NOTE: For full coverage the library must be built with `make STATIC=0 SYM=1` (or `DBG=1`) so that symbol information is present.

`tool_getenvvars.sh`

Scans the LIBXSTREAM source tree (`src/*.c`) for calls to `getenv` and prints a sorted, deduplicated list of every environment variable used at runtime, separated into LIBXSTREAM-specific (`LIBXSTREAM_*`) and other variables.

`scripts/tool_getenvvars.sh`

`tool_openccl.sh`

Converts OpenCL kernel files (`.cl`) — and optionally CSV parameter files — into a C header whose string literals can be compiled straight into the host binary. Include guards are stripped by default, and `#include` directives inside kernels are recursively resolved (inline).

`scripts/tool_openccl.sh [options] infile.cl [infile2.cl ..] [infile.csv ..] outfile.h`

	Flag		Description				<code>-k, --keep</code>		Keep include guards (normally stripped)			<code>-b N, --banner N</code>		Copy the first N lines of the first <code>.cl</code> file as a banner			<code>-p DIR, --params DIR</code>		Directory containing CSV parameter files			<code>-c, -d, --debug, --comments</code>		Preserve comments in the generated source			<code>-v, --verbose</code>		Echo the command line		
--	------	--	-------------	--	--	--	-------------------------	--	---	--	--	-------------------------------	--	---	--	--	-----------------------------------	--	--	--	--	--	--	---	--	--	----------------------------	--	-----------------------	--	--

tool_version.sh

Derives the project version from Git tags and revision count. It can emit a plain version string, individual version components, or a complete C header with `#define` guards.

```
## full version string (branch-tag-patch)
scripts/tool_version.sh

## generate a C version header with a given symbol prefix
scripts/tool_version.sh LIBXSTREAM -1

## single component: 1=major, 2=minor, 3=update, 4+=patch
scripts/tool_version.sh LIBXSTREAM 2
```

| Argument | Description | ||| | PREFIX | Symbol prefix used in the generated header (uppercased) | | CMPNT | 0 (default)
— version string; negative — C header; 1–3 — major/minor/update; >3 — patch count | | SHIFT | Added to the patch
count (default: 0) |