Trait CompressedCapability

Trait rust_cheri_compressed_cap::CompressedCapability

```
pub trait CompressedCapability: Sized + Copy + Clone {
    type Length: NumType + From<Self::Addr>;
    type Offset: NumType + From<Self::Addr>;
    type Addr: NumType + Into<Self::Offset> + Into<Self::Length>;
    type FfiLength: FfiNumType<Self::Length>;
    type FfiOffset: FfiNumType<Self::Offset>;
[+] Show associated constants and methods
}
```

Trait defining an Rust version of the public API for a specific capability type. A type X implementing CompressedCapability is equivalent to the API provided by `cheri_compressed_cap_X.h` in C, where `ccx_cap_t` is equivalent to CcxCap.

It is not recommended to call the trait functions directly. Instead, use one of the crate::wrappers.

Associated Types

`type Length: NumType + From<Self::Addr>` [src]

ccx_length_t Rust-land equivalent - should be a superset of Addr

`type Offset: NumType + From<Self::Addr>` [src]

ccx_offset_t Rust-land equivalent - should be a superset of Addr

`type Addr: NumType + Into<Self::Offset> + Into<Self::Length>` [src]

ccx_addr_t equivalent

`type FfiLength: FfiNumType<Self::Length>` [src]

ccx_length_t C-land equivalent - should have a memory layout identical to the C ccx_length_t. This is separate from Length because for 128-bit types the Rust and C versions may not look the same. In practice, we just assume they are the same (see crate::c_funcs documentation).

`type FfiOffset: FfiNumType<Self::Offset>` [src]

ccx_offset_t C-land equivalent - should have a memory layout identical to the C ccx_offset_t. See Self::FfiLength for an explanation.

Associated Constants

`const PERM_GLOBAL: u32` [src]

CCX_PERM_GLOBAL equivalent These are the same for 64 and 128bit, but should be overridden for Morello-128

`const PERM_EXECUTE: u32` [src]

`const PERM_LOAD: u32` [src]

`const PERM_STORE: u32` [src]

`const PERM_LOAD_CAP: u32` [src]

`const PERM_STORE_CAP: u32` [src]

`const PERM_STORE_LOCAL: u32` [src]

`const PERM_SEAL: u32` [src]

`const PERM_CINVOKE: u32` [src]

`const PERM_UNSEAL: u32` [src]

`const PERM_ACCESS_SYS_REGS: u32` [src]

`const PERM_SETCID: u32` [src]

`const MAX_REPRESENTABLE_OTYPE: u32` [src]

`const OTYPE_UNSEALED: u32` [src]

CCX_OTYPE_UNSEALED equivalent

`const OTYPE_SENTRY: u32` [src]

`const OTYPE_RESERVED2: u32` [src]

`const OTYPE_RESERVED3: u32` [src]

`const MAX_UNRESERVED_OTYPE: u32` [src]

Required methods

`fn compress_raw(src_cap: &CcxCap<Self>) -> Self::Addr` [src]

Generate the `pesbt` bits for a capability (the top bits, which encode permissions, object type, compressed bounds, etc.) This transformation can be undone with

Self::decompress_raw.

This is presumably intended for storing compressed capabilities in e.g. registers. Its counterpart for storing compressed capabilities in memory is Self::compress_mem.

```
fn decompress_raw(                                        [src]
    pesbt: Self::Addr,
    cursor: Self::Addr,
    tag: bool
) -> CcxCap<Self>
```

Decompress a (pesbt, cursor) pair into a capability. This transformation can be undone with Self::compress_raw.

```
fn compress_mem(src_cap: &CcxCap<Self>) -> Self::Addr     [src]
```

Generate the `pesbt` bits for a capability (the top bits, which encode permissions, object type, compressed bounds, etc.) This transformation can be undone with Self::decompress_mem.

This is presumably intended for storing compressed capabilities in memory. It is equivalent to calling Self::compress_raw and XOR-ing the result with a “null mask”. Presumably this transformation prevents all-zero data from being interpreted as a capability?

```
fn decompress_mem(                                        [src]
    pesbt: Self::Addr,
    cursor: Self::Addr,
    tag: bool
) -> CcxCap<Self>
```

Decompress a (pesbt, cursor) pair into a capability. This transformation can be undone with Self::compress_mem.

This is equivalent to XOR-ing the pesbt with a “null mask” and calling Self::decompress_raw. Presumably the null mask prevents all-zero data from being interpreted as a capability?

```
fn get_uperms(cap: &CcxCap<Self>) -> u32                  [src]
```

Gets the user/software-defined permissions from the CcxCap::cr_pesbt field

Counterpart: Self::update_uperms

```
fn get_perms(cap: &CcxCap<Self>) -> u32                   [src]
```

Gets the hardware-defined permissions from the CcxCap::cr_pesbt field

Counterpart: Self::update_perms

```
fn get_otype(cap: &CcxCap<Self>) -> u32                   [src]
```

Gets the object type from the CcxCap::cr_pesbt field

Counterpart: Self::update_otype

```
fn get_reserved(cap: &CcxCap<Self>) -> u8
```

[src]

Gets the reserved bits from the CcxCap::cr_pesbt field

Counterpart: Self::update_reserved

```
fn get_flags(cap: &CcxCap<Self>) -> u8
```

[src]

Gets the flags from the CcxCap::cr_pesbt field

Counterpart: Self::update_flags

```
fn update_uperms(cap: &mut CcxCap<Self>, value: u32)
```

[src]

Updates the user/software-defined permissions field in CcxCap::cr_pesbt

Counterpart: Self::get_uperms

```
fn update_perms(cap: &mut CcxCap<Self>, value: u32)
```

[src]

Updates the hardware-defined permissions field in CcxCap::cr_pesbt

Counterpart: Self::get_perms

```
fn update_otype(cap: &mut CcxCap<Self>, value: u32)
```

[src]

Updates the object type field in CcxCap::cr_pesbt

Counterpart: Self::get_otype

```
fn update_reserved(cap: &mut CcxCap<Self>, value: u8)
```

[src]

Updates the reserved field in CcxCap::cr_pesbt

Counterpart: Self::get_reserved

```
fn update_flags(cap: &mut CcxCap<Self>, value: u8)
```

[src]

Updates the flags field in CcxCap::cr_pesbt

Counterpart: Self::get_flags

```
fn extract_bounds_bits(pesbt: Self::Addr) -> CcxBoundsBits
```

[src]

Extracts the floating-point encoded bounds from CcxCap::cr_pesbt

```
fn set_bounds(
    cap: &mut CcxCap<Self>,
    req_base: Self::Addr,
    req_top: Self::Length
) -> bool
```

[src]

Sets the capability bounds to bounds that encompass (req_base, req_top). Because a floating-point representation is used for bounds, it may not be able to set (req_base, req_top) exactly. In this case it will return False.

Updates CcxCap::cr_pesbt, CcxCap::_cr_top, CcxCap::cr_base

```
fn is_representable_cap_exact(cap: &CcxCap<Self>) -> bool          [src]
```

Check if the range (CcxCap::cr_base, CcxCap::_cr_top) can be encoded exactly with the floating-point encoding

```
fn is_representable_new_addr(                                       [src]
    sealed: bool,
    base: Self::Addr,
    length: Self::Length,
    cursor: Self::Addr,
    new_cursor: Self::Addr
) -> bool
```

Check if a capability with the parameters `sealed, base, length, cursor` would be representable if the cursor were updated to `new_cursor`.

```
fn make_max_perms_cap(                                              [src]
    base: Self::Addr,
    cursor: Self::Addr,
    top: Self::Length
) -> CcxCap<Self>
```

Generate a capability for `base, top, cursor` with the maximum available permissions

```
fn get_representable_length(length: Self::Length) ->               [src]
Self::Length
```

Get the minimum representable length greater than or equal to `length`.

If `get_representable_length(l) == l` then bounds of length `l` are exactly representable (if properly aligned).

See also Self::get_required_alignment, Self::get_alignment_mask.

```
fn get_required_alignment(length: Self::Length) -> Self::Length    [src]
```

Get the alignment required for bounds of some `length` to be exactly represented.

See also Self::get_representable_length, Self::get_alignment_mask.

```
fn get_alignment_mask(length: Self::Length) -> Self::Length        [src]
```

Get a mask which aligns a bounds of some `length` to be exactly representable.

See also Self::get_representable_length, Self::get_required_alignment.

Implementors

```
mpl CompressedCapability for Cc64                                    [src]

    type Length = u64

    type Offset = i64

    type Addr = u32

    type FfiLength = u64

    type FfiOffset = i64

    const MAX_REPRESENTABLE_OTYPE: u32                               [src]
```

```
_CC_N(OTYPE_UNSEALED_SIGNED) = (((int64_t)-1) - 0u)```
The OTYPE field is 4 bits (50:47) in CC64
```

```
[−] impl CompressedCapability for Cc128                              [src]

    type Length = u128

    type Offset = i128

    type Addr = u64

    type FfiLength = u128

    type FfiOffset = i128

    const MAX_REPRESENTABLE_OTYPE: u32                               [src]
```

The OTYPE field is 18 bits (108:91) in CC128