Application Note: AN10082

How to use unsafe pointers

This application note is a short how-to on programming/using the xTIMEcomposer tools. It shows how to use unsafe pointers.

Required tools and libraries

This application note is based on the following components:

- xTIMEcomposer Tools - Version 14.0.0

Required hardware

Programming how-tos are generally not specific to any particular hardware and can usually run on all XMOS devices. See the contents of the note for full details.

1 How to use unsafe pointers

An unsafe pointer type is provided for compatibility with C and to implement dynamic, aliasing data structures (for example linked lists). This is not the default pointer type and the onus is on the programmer to ensure memory safety for these types.

An unsafe pointer is opaque unless accessed in an unsafe region. A function can be marked as unsafe to show that its body is an unsafe region:

```
unsafe void f(int * unsafe x) {
  // We can dereference x in here,
  // but be careful - it may point to garbage
  printintln(*x);
}
```

Unsafe functions can only be called from unsafe regions. You can make a local unsafe region by marking a compound statement as unsafe:

```
void g(int * unsafe p) {
  int i = 99;
  unsafe {
    p = &i;
    f(p);
  }
  // Cannot dereference p or call f from here
}
```

These regions allow the programmer to manage the parts of their program that are safe by construction and the parts that require the programmer to ensure safety.

Within unsafe regions, unsafe pointers can be explicitly cast to safe pointers - providing a contract from the programmer that the pointer can be regarded as safe from then on.

It is undefined behavior for an unsafe pointer to be written from one task and read from another.