Application Note: AN10063

How to use restricted pointers

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

If you declare a global pointer, it defaults to an *restricted* pointer.

```
int x[10] = {1,2,3,4,5,6,7,8,9,10};
int *p = &x[5];
```

To maintain safety, global pointers of this kind have restricted use. In particular, they cannot be assigned to point to a different object once they have been initialized. In this example, the pointer p can only ever point to x.

The pointer p provides a non-aliased reference to x. Initializing it to point at x means that x cannot be directly accessed in the program.

Function parameters also default to this *restricted* kind of pointer from the point of view of the caller. So if a function takes an argument `int *y` - that parameter can be assumed to point to an object that nothing else points to. Within the function body, however, function arguments are treated like local aliasing pointers (so can be assigned to).

The restrictions on global pointers and parameters are to ensure memory safety as pointers are passed between parts of your program. Local pointers do not have these restrictions. To allow more flexible pointers at a global level (or when passing between functions), the pointer types need to be annotated with a pointer kind indicating its use (e.g. movable or unsafe).

Restricted pointers can be dereferenced like a normal C pointer.

```
void f(int *y) {
  printf("The value pointed to is: %d\n", y[1]);
}

int main() {
  printf("The value pointed to is: %d\n", *p);
  f(p+3);
  return 0;
}
```