Application Note: AN10033

How to use movable pointers

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

Movable pointers are pointers that only exist in one variable at a time. The initialization of a movable pointer is special. At initialization time you can set the pointer to point to a particular object. As with global pointers, the object pointed to is then hidden so it cannot be accessed directly during the lifetime of the pointer:

```
void g()
{
  int * movable x = &y[0];   // x is initialized to point to y

  // cannot access y during the lifetime of x
  ...
}
```

Movable pointers cannot be copied. For example the following code is invalid:

```
int * movable y;
...
void g()
{
  int * movable x = &a[0];
  y = x;  // This is an error, you cannot copy the value of x.
  ...
}
```

Instead of copying, the pointers can be “moved” using the special move operator. The move operator transfers the pointer value and sets the original pointer to null:

```
void f() {
  int * movable y;
  int a[5] = {4,5,6,7,8};
  int * movable x = &a[0];
  y = move(x);

  if (x == null)
    printf("x is now null.\n");

  printf("The contents of y is %d\n", *y);

  x = move(y);
}
```

When a pointer is moved the original pointer is set to null. So the pointer only exists in one variable.

To avoid dangling pointers, movable pointers must have their original value when they go out of scope (so the pointer is not left dangling anywhere else):

```
int * movable y;
..
void g()
{
int * movable x = &a[0];
y = move(x);
...
x = move(y);
...
} <--- x must have its original value at this point
```

A runtime check enforces this restriction.

As a result of this, movable pointers need to be moved back into their original variable location before that variable goes out of scope.

The move operator can also be used to move a pointer into or out of a function :

```
int * movable p_global;

void grab_pointer(int * movable x) {
  // store x in a global
  p_global = move(x);
}

int * movable retreive_pointer(void) {
  // retreive x from a global
  *p_global += 1;
  return move(p_global);
}

void g(void) {
  int i = 77;
  int * movable x = &i;
  grab_pointer(move(x));
  if (x == null)
    printf("x is now null.\n");
  x = retreive_pointer();
  printf("Contents of x is %d\n", *x);
}
```