Application Note: AN10043

How to use a par statement

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

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 a par statement

xC programs are built of tasks that run in parallel. Tasks are just code so you can define them like normal C functions.

```
void task1(int x)
{
  printf("Hello world - %d\n", x);
}
```

Tasks can take any arguments but generally have no return value. Often tasks do not return at all and consist of a never ending loop:

```
void task1(args) {
     ... initialization ...
     while (1) {
        ... main loop ...
     }
}
```

Tasks can be run in parallel from any function. However, it is only in the function `main` that tasks can be set up to run on multiple different xCORE tiles.

Running tasks in parallel is done with the `par` construct (short for “run in par-allel”). Here is an example `par` statement

```
par {
  task1(5);
  task2();
}
```

This statement will run `task1` and `task2` in parallel to completion. It will wait for both tasks to complete before carrying on.

Tasks run on separate logical cores run in parallel in the hardware so there is no notion of priority or scheduling between the tasks.

This example uses `printf` which sends output over the debug adaptor to the console. The C standard `printf` routine takes up quite a bit of memory. Other debug printing libraries are available with a smaller memory footprint e.g. those found in the `print.h` header supplied with the development tools or the logging module xSOFTip library.