Application Note: AN10038

How to nest combinable function calls

This application note is a short how-to on programming/using the xTIMEcomposer tools. It shows how to nest combinable function calls.

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 nest combinable function calls

Suppose you have two combinable functions:

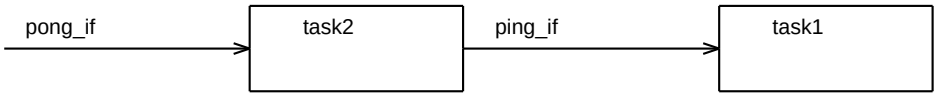
```
interface ping_if {
  void ping();
};

interface pong_if {
  void pong();
};

[[combinable]]
void task1(server interface ping_if i)
{
  while(1) {
    select {
    case i.ping():
      printf("Task1 received a ping!\n");
      break;
    }
  }
}

[[combinable]]
void task2(server interface pong_if i_pong, client interface ping_if i_ping)
{
  while (1) {
    select {
    case i_pong.pong():
      i_ping.ping();
      break;
    }
  }
}
```

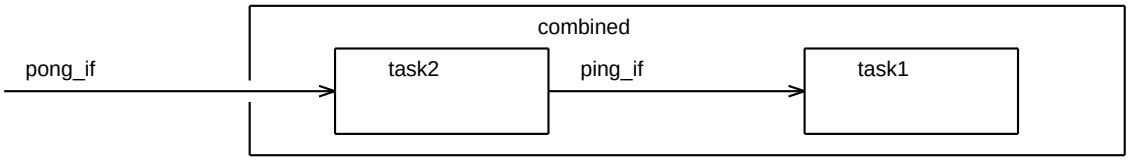
Sometimes, functions are always expected to be connected and combined together. So task1 and task2 are always supposed to be connected:



It is possible to create a new function definition that contains a combined par statement consisting of these two tasks:

```
[[combinable]]
void combined(server interface pong_if i_pong)
{
  interface ping_if i_ping;
  [[combine]]
  par {
    task1(i_ping);
    task2(i_pong, i_ping);
  }
}
```

This groups the tasks together:



It is then possible to combine this new task with other combinable tasks:

```
[[combinable]] void task3(client interface pong_if i);

int main() {
  interface pong_if i;
  par {
    on tile[0].core[0]: combined(i);
    on tile[0].core[0]: task3(i);
  }
  return 0;
}
```