UNRAVELLING XC CONCEPTS

[[COMBINE]], [[COMBINABLE]], [[DISTRIBUTE]], [[DISTRIBUTABLE]] AND [[DISTRIBUTED(..)]] PLUS PAR AND ON..

ØYVIND TEIG[2]
CODER AND BLOGGER

CPA 2018 IN DRESDEN, GERMANY. AUGUST 19–22, 2018
COMMUNICATING PROCESS ARCHITECTURES[1]
A FRINGE LECTURE

my 23rd WoTUG/CPA

[1] http://wotug.cs.unlv.edu

[2] http://www.teigfam.net/oyvind/home/ = *Aclassifier* with 40 years of embedded and safety-critical coding (including some years of occam). Home lab («retired») since June 2017

20Aug2018

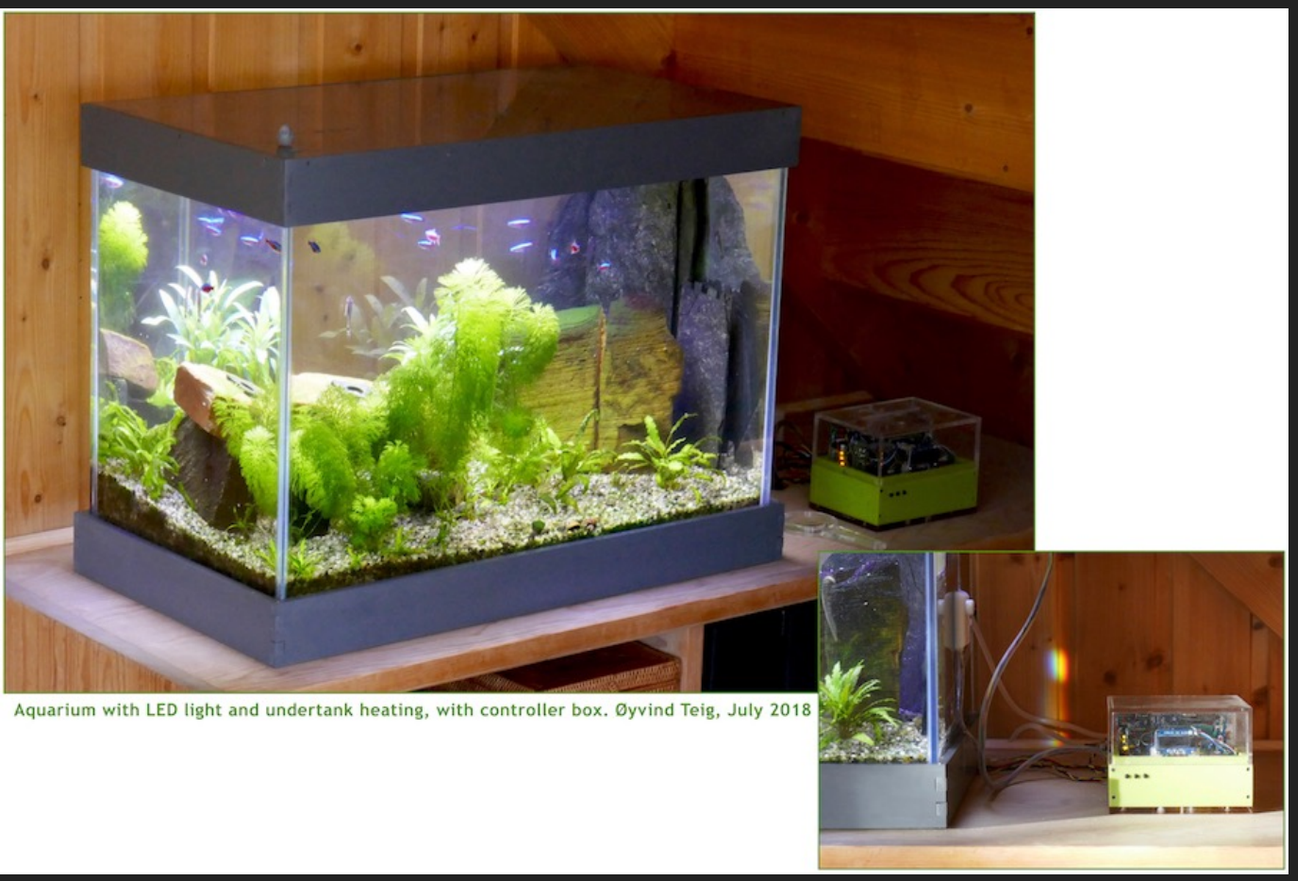
DISCLAIMER:

I HAVE NO OTHER ASSOCIATION WITH XMOS THAN I USED TO HAVE WITH INMOS: `null`,

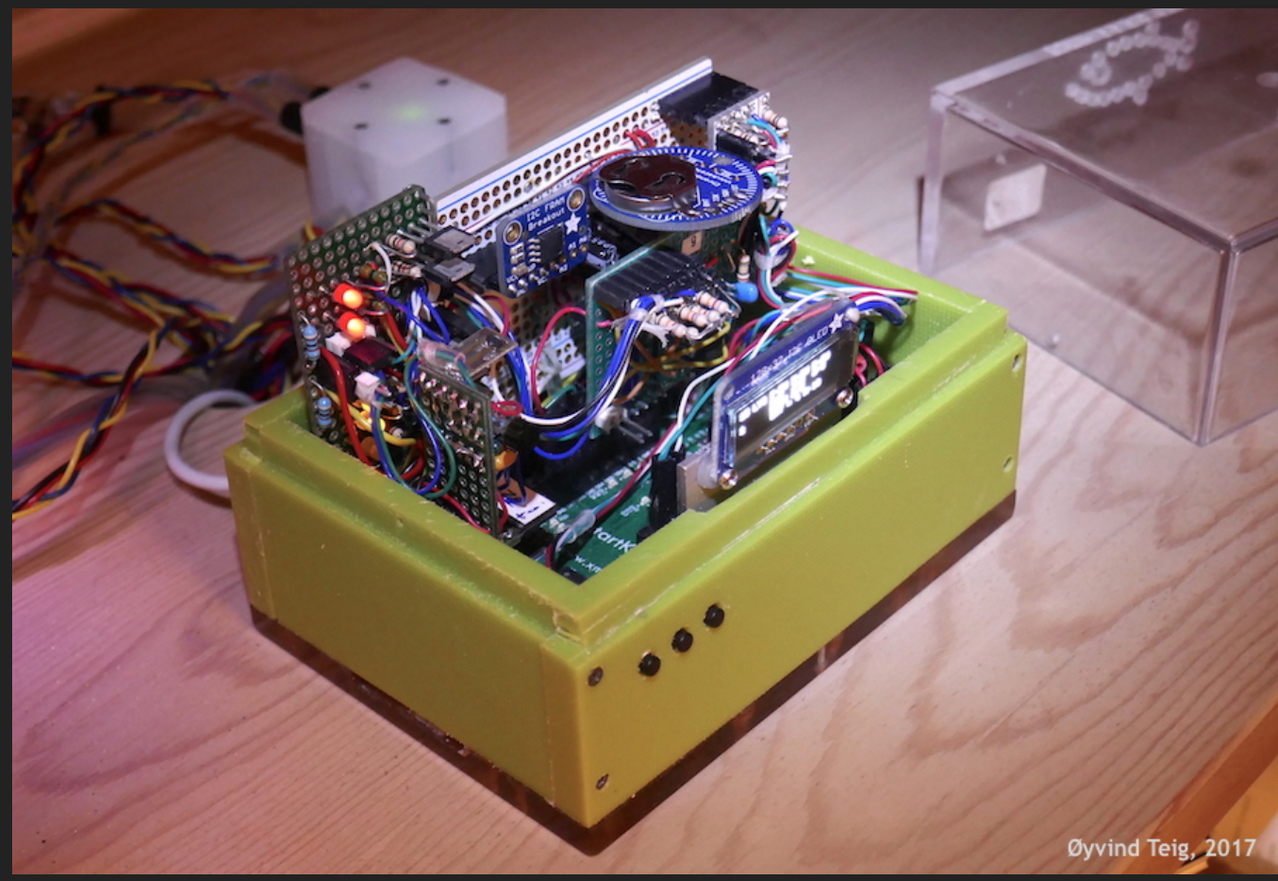
EXCEPT THE OCCASIONAL CONTACT THROUGH MAIL OR THE XCORE FORUM,

ie NO MONEY, NO GIFTS, NO ADS, NO NOTHING..

..ONLY FUN AND EXPENSES



Aquarium with LED light and undertank heating, with controller box. Øyvind Teig, July 2018



Øyvind Teig, 2017

BUT I DO USE XMOS BOARDS IN MY RATHER EXTENDED AQUARIUM PROJECT

AND AS A BACKGROUND FOR MY XMOS RELATED BLOGGING (SAME DISCLAIMER).

LIKE IN THE NOTE « XC IS C PLUS X»[3]

---

[3] http://www.teigfam.net/oyvind/home/technology/141-xc-is-c-plus-x/

C PLUS ‘X’?!

‘X’ THE UNKNOWN, NO QUESTION-MARK. BUT FUN

Communicating Process Architectures 2018
K. Chalmers, J.B. Pedersen et al. (Eds.)
IOS Press, 2018

Unravelling `XC` concepts `[[combine]]`, `[[combinable]]`, `[[distribute]]`, `[[distributable]]` and `[[distributed(..)]]` plus `par` and `on`..

In search of understanding how some attributes, as present in XC code, by the compiler are treated weaker and more general and ultimately may be ignored altogether - in the runnable code for xCore multicore microcontrollers

Øyvind TEIG[1]
Coder and blogger

The XC programming language is designed to make runnable multitask programs for XMOS' xCore multicore microcontrollers, spread on *tiles* and *cores*. This presentation addresses only a few aspects of XC. What is a *combinable* and what is a *distributable* task, syntactically and semantically, but also resource wise and geographically on the chip? The xCore compiler handles the lowering of interfaces onto statically and dynamically allocated channel resources. The microcode and built-in scheduler also reflect them. Of the rather limited amount of resources, one must make it with 32 *chanends* per tile, with seemingly loose coupling between the XC code and the final number of chanends. Fiddling around, rather

can of reserved words and observe what might possibly happen. Like, there must be an electric motor inside, because the sound of it is so pleasant.

optimisation, channel, interface, critical region, lock, multicore, scheduler, scheduling.

[1] http://www.teigfam.net/oyvind/home/ and http://www.teigfam.net/oyvind/work/work.html, embedded and safety-critical coder, retired since June 2017

```
interface button_if_t {
    void but (int x);
};

typedef enum {false,true} bool;

[[distributable]] // [[combinable]]
void handle (server interface button_if_t i_but[3]) {
    // int cnt = 0;
    // timer tmr;
    // int   time;
    // bool  timeout = false;
    // tmr :> time;
    while (1) {
        select {
            case i_but[int i].but (int ms) : {
                // Do something
                // timeout = false;
                break;
            }
            // case tmr when timerafter(time) :> void: {
            //     timeout = true;
            //     time += XS1_TIMER_HZ; // One second
            //     break;
            // }
        }
        // cnt++;
    }
}

int main (void) {
    interface button_if_t i_but[3];
    par {
        [[combine]]                 [[combine]]
        par {                       par (int j = 0; j < 3; j++) {
            handle (i_but);             button (i_but[j]);
            button (i_but[0]);      }
            button (i_but[1]);      [[distribute]] // [[combine]]
            button (i_but[2]);      par {
        }                               handle (i_but);
                                    }
    }
    return 0;
}
```

XC

1

```
Constraint check for tile[0]:
  Cores available:            8,   used:          4 .  OKAY
  Timers available:          10,   used:          4 .  OKAY
  Chanends available:        32,   used:          6 .  OKAY
  Memory available:       65536,   used:       1464 .  OKAY
    (Stack: 372, Code: 882, Data: 210)
```

2

3

```
Constraint check for tile[0]:
  Cores available:            8,   used:          1 .  OKAY
  Timers available:          10,   used:          1 .  OKAY
  Chanends available:        32,   used:          0 .  OKAY
  Memory available:       65536,   used:       1852 .  OKAY
    (Stack: 404, Code: 1228, Data: 220)
Constraints checks PASSED.
```

4

```
Constraint check for tile[0]:
  Cores available:            8,   used:          1 .  OKAY
  Timers available:          10,   used:          1 .  OKAY
  Chanends available:        32,   used:          0 .  OKAY
  Memory available:       65536,   used:       1756 .  OKAY
    (Stack: 404, Code: 1132, Data: 220)
Constraints checks PASSED.
```

5

```
Constraint check for tile[0]:
  Cores available:            8,   used:          2 .  OKAY
  Timers available:          10,   used:          2 .  OKAY
  Chanends available:        32,   used:          4 .  OKAY
  Memory available:       65536,   used:       1728 .  OKAY
    (Stack: 376, Code: 1090, Data: 262)
Constraints checks PASSED.
```

6 Wrong error message

```
../src/main.xc:366:1: error: distributed statement must be a call
                             to a distributable function
```

MY XCORE-200 EXPLORERKIT BOARDS' PROCESSOR

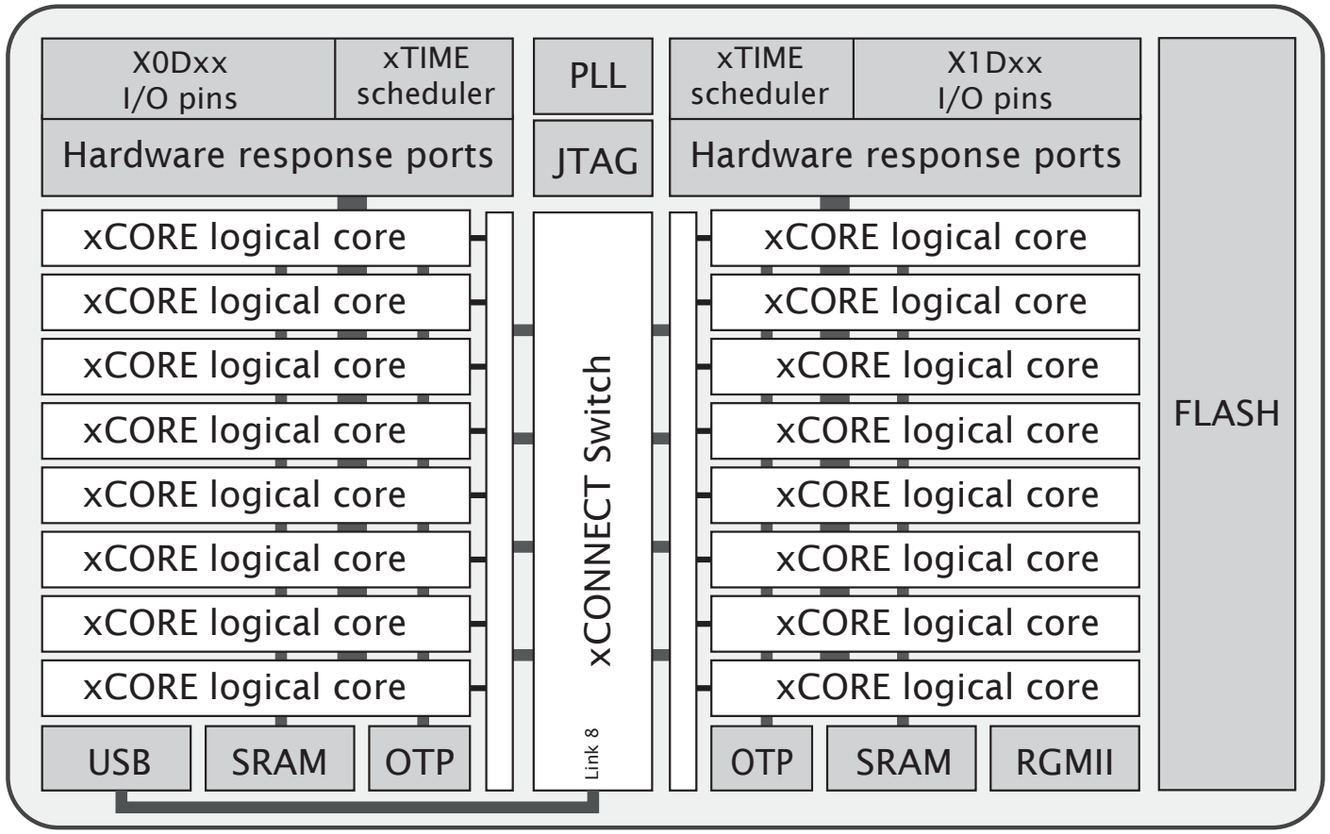


Figure 1: XEF216-512-TQ128 block diagram, from **XEF216-512-TQ128 Datasheet**. 2018/03/23
Document Number: X006990
http://www.xmos.com/download/private/XEF216-512-TQ128-Datasheet%281.15%29.pdf.
As used in the xCORE-200 eXplorerKIT.

- 2 tiles (500 MIPS per tile (or dual))
- 8 cores per tile (=«Logical cores»)
- xTIME scheduler. If # cores active:
  - 1-4 cores: 1/4 cycles each
  - 5-8 cores: all cycles shared out
  - Deterministic thread execution
  - Thread safe
  - pragma for some deadlines
- Channels: untyped. Synch or asynch
  - XC `chanends` (32 per tile)
  - Not between tasks on the same core
- XC `interface` (typed and role/session)
  - *May* use `chanends` or locks or sharing of `select` or context (blocks of state data)
- Shared memory & no data bus contention
  - No cache
  - No DMA
  - I/O does not use memory bus
- Also supported/used by XC
  - Locks (4 per tile). Runtime
  - I/O ports
  - Clock blocks (6 per tile)
  - Timers (10 pr tile)

XC real life @ Aquarium controller
On XMOS startKIT (I have several, after eol reached)

```
int main() {
    ...  chan and interfaces
    par {
        on tile[0]: installExceptionHandler();
        par {
                        startkit_adc         (c_analogue);                                          // Is none since a "service"/hardware
            on tile[0]: My_startKIT_ADC_Task (i_startkit_adc_acquire, i_lib_startkit_adc_commands,   // Is none since contains a nested select
                                              NUM_STARTKIT_ADC_NEEDED_DATA_SETS);
            on tile[0]: System_Task          (i_i2c_internal_commands[0], i_i2c_external_commands[0], // Is none since contains a nested select
                                              i_lib_startkit_adc_commands[0], i_port_heat_light_commands[0],
                                              i_temperature_heater_commands[0], i_temperature_water_commands,
            /* tile[0].core[0] not now */     i_buttons, i_irq, i_radio);
            on tile[0]: adc_task             (i_startkit_adc_acquire, c_analogue,                   // [[combinable]]
                                              ADC_PERIOD_TIME_USEC_ZERO_IS_ONY_QUERY_BASED);
        }
        on tile[0]: {
            [[combine]]
            par {
                Button_Task (IOF_BUTTON_LEFT,   inP_button_left,   i_buttons[IOF_BUTTON_LEFT]);   // [[combinable]]
                Button_Task (IOF_BUTTON_CENTER, inP_button_center, i_buttons[IOF_BUTTON_CENTER]); // [[combinable]]
                Button_Task (IOF_BUTTON_RIGHT,  inP_button_right,  i_buttons[IOF_BUTTON_RIGHT]);  // [[combinable]]
            }
        }
        on tile[0]: {
            [[combine]]
            par {
                I2C_Internal_Task         (i_i2c_internal_commands);      // [[combinable]]
                I2C_External_Task         (i_i2c_external_commands);      // [[distributable]]
                Temperature_Heater_Task   (i_temperature_heater_commands, // [[combinable]]
                                           i_i2c_external_commands[1],
                                           i_port_heat_light_commands[1]);
                Temperature_Water_Task    (i_temperature_water_commands,  // [[combinable]]
                                           i_temperature_heater_commands[1]);
                Port_Pins_Heat_Light_Task (i_port_heat_light_commands);   // [[combinable]]
            }
        }
        on tile[0]: { // To avoid Error: lower bound could not be calculated (xTIMEcomposer 14.3.3)
            [[combine]]
            par {
                RFM69_driver    (i_radio, p_spi_aux, i_spi[SPI_CLIENT_0], SPI_CLIENT_0);             // [[distributable]]
                spi_master_2    (i_spi, NUM_SPI_CLIENT_USERS, p_sclk, p_mosi, p_miso,                // [[distributable]]
                                 SPI_CLOCK, p_spi_cs_en, maskof_spi_and_probe_pins, NUM_SPI_CS_SETS);
                IRQ_detect_task (i_irq, p_spi_irq, probe_config, null, 0);                           // [[combinable]]
            }
        }
    }
    return 0;
}
```

```
...  chan and interfaces
   // interfaces:
   button_if                     i_buttons[3];
   spi_master_if                 i_spi    [1];
   radio_if_t                    i_radio;
   irq_if_t                      i_irq;
   i2c_external_commands_if      i_i2c_external_commands [2];
   i2c_internal_commands_if      i_i2c_internal_commands [1];
   startkit_adc_acquire_if       i_startkit_adc_acquire;
   lib_startkit_adc_commands_if  i_lib_startkit_adc_commands  [1];
   port_heat_light_commands_if   i_port_heat_light_commands   [2];
   temperature_heater_commands_if i_temperature_heater_commands[2];
   temperature_water_commands_if  i_temperature_water_commands;
```

```
Constraint check for tile[0]:
  Cores available:            8,   used:          7 .  OKAY
  Timers available:          10,   used:          8 .  OKAY
  Chanends available:        32,   used:         26 .  OKAY
  Memory available:       65536,   used:      61620 .  OKAY
    (Stack: 6472, Code: 49386, Data: 5762)
Constraints checks PASSED.
```

- Common term for `[[combine]]`, `[[combinable]]`, `[[distribute]]`, `[[distributable]]` and `[[distributed(..)]]` does not seem to exist
- Not all shown here. More on «XC is C plus X»
- `chan` and `interface` code compared side by side (next page, small text, just for reference) (It also shows where the tool fails for some usages)
- The XC reference I use the most[1]
- Plus the XCORE Exchange community forum[2]

---

[1] **XMOS Programming Guide**
as of 14Aug2018: 2015/9/21 version F, 2015/9/18 in the document,
https://www.xmos.com/published/xmos-programming-guide

[2] https://www.xcore.com

XC code, see http://www.teigfam.net/oyvind/home/technology/141-xc-is-c-plus-x/

XC is C plus X, The combined code: 6 to zero channels by Øyvind Teig (6Jul2018)

```
#if defined TEST_CHAN_AND_COMBINE_TEST

    #include <platform.h>
    #include <stdio.h>
    #include <timer.h> // XS1_TIMER_HZ etc

    #define DEBUG_PRINT_TEST 0
    #if (DEBUG_PRINT_TEST == 1)
        // Uses 1 timer and one chanend (not counted below)
        #define debug_print(fmt, ...) do \
            { if(DEBUG_PRINT_TEST) printf(fmt, __VA_ARGS__); } while (0)
    #else
        #define debug_print(fmt, ...)
    #endif
```

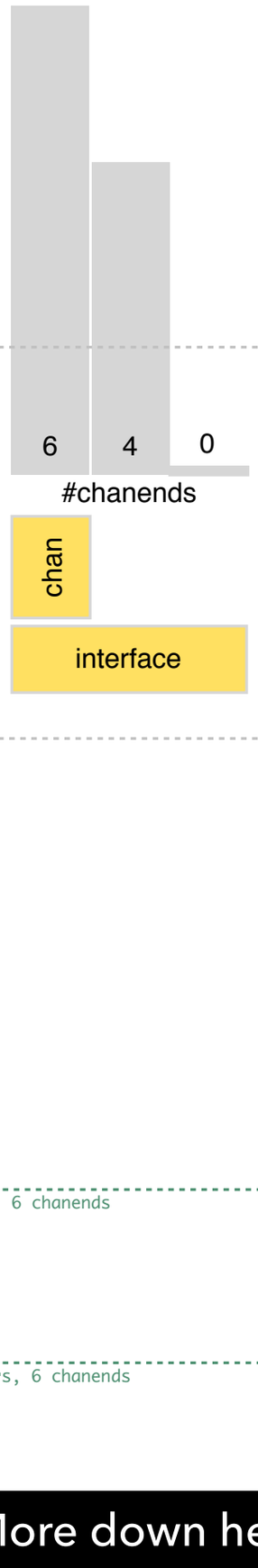
chan

```
    [[combinable]]
    void button (chanend c_out) {
        timer t;
        int s;
        t :> s;
        while (1) {
            select {
                case t when timerafter(s) :> void: {
                    c_out <: (s/XS1_TIMER_KHZ); // ms
                    s += XS1_TIMER_HZ;
                    break;
                }
            }
        }
```

```
[[combinable]]
void handle (chanend c_but[3]) {
    int val;
    while (1) {
        select {
            case c_but[int i] :> val: {
                debug_print ("handle: from %d val %u\n", i, val);
                break;
            }
        }
    }
}

#define DO_PLACED 1 // 1-4 works

int main (void) {
    chan c_but[3]; // Using 6 chanends always
    par {
        #if (DO_PLACED == 1) // Works, also with interface. Uses 4 cores, 4 timers, 6 chanends
            on tile[0].core[0]: handle (c_but);
            par {
                on tile[0].core[2]: button (c_but[0]);
                on tile[0].core[3]: button (c_but[1]);
                on tile[0].core[4]: button (c_but[2]);
            }
        #elif (DO_PLACED == 2) // Works, also with interface. Uses 2 cores, 2 timers, 6 chanends
            on tile[0].core[0]: handle (c_but);
            par {
                on tile[0].core[1]: button (c_but[0]);
                on tile[0].core[1]: button (c_but[1]);
                on tile[0].core[1]: button (c_but[2]);
            }
        #elif (DO_PLACED == 3) // Works, also with interface. Uses 4 cores, 4 t
```




```
#elif defined TEST_INTERFACE_AND_COMBINE_TEST

    #include <platform.h>
    #include <stdio.h>
    #include <timer.h> // XS1_TIMER_HZ etc

    #define DEBUG_PRINT_TEST 1
    #if (DEBUG_PRINT_TEST == 1)
        // Uses 1 timer and one chanend (not counted below)
        #define debug_print(fmt, ...) do \
            { if(DEBUG_PRINT_TEST) printf(fmt, __VA_ARGS__); } while (0)
    #else
        #define debug_print(fmt, ...)
    #endif
```

interface

```
    interface ifa {
        void but (int x);
    };
```

```
    [[combinable]]
    void button (client interface ifa i_but) {
       timer t;
       int s;
       t :> s;
       while (1) {
           select {
               case t when timerafter(s) :> void: {
                   i_but.but(s/XS1_TIMER_KHZ); // ms
                   s += XS1_TIMER_HZ;
                   break;
               }
           }
       }
```

```
[[combinable]]
void handle (server interface ifa i_but[3]) {
    while (1) {
        select {
            case i_but[int i].but (int val) : {
                debug_print ("handle: from %d val %u\n", i, val);
                break;
            }
        }
    }
}

#define DO_PLACED 6 // 1-6 works


int main (void) {
    interface ifa i_but[3]; // 6 to zero chanends
    par {
        #if (DO_PLACED == 1) // Works, also with chan. Uses 4 cores, 4 timers, 6 chanends
            on tile[0].core[0]: handle (i_but);
            par {
                on tile[0].core[2]: button (i_but[0]);
                on tile[0].core[3]: button (i_but[1]);
                on tile[0].core[4]: button (i_but[2]);
            }
        #elif (DO_PLACED == 2) // Works, also with chan. Uses 2 cores, 2 timers, 4 chanends
            on tile[0].core[0]: handle (i_but);
            par {
                on tile[0].core[1]: button (i_but[0]);
                on tile[0].core[1]: button (i_but[1]);
                on tile[0].core[1]: button (i_but[2]);
```

More down here in blog note

```
        #elif (DO_PLACED == 3) // Works, also with chan. Uses 4 cores, 4 timers, 6 chanends
```

INSIDE THE TOOL CHAIN (FROM AN INSIDER)

- The xCore compiler handles the «lowering of interfaces» onto statically and dynamically allocated channel resources
- Program Content Analysis (optional but on by default) into a pca-file (xml)
- Compilation into Abstract Syntax Tree
  - Specialisation stage using pca-file
  - The XC compiler will generate multiple versions of «interface lowered» code
    - for when the server and client are on different tiles or cores
    - for when the server and client are actually combined
    - for when the server and client are actually distributed
    - for when a server may need to be re-entrant (yielding), due to a possible calling cycle
- The linker runs, linking together the object code, and throwing away unused (non specialised) functions
- In an .s-file there would be duplicate content but with different boiler plating regarding how chanends and blocks of state data (holding chanends) are used

SUMMARY

- These terms are much «trial and see» (how the sw or tool works). Maybe this has to be so?
- The tool goes to extremes, with free will - guided by me, to build thread safe and smart code to utilise the cores and not make any task interfere with any other task
- Here → is my aquarium LED light PWM task that watches over light intensity. It flickered only before the algorithm was finished
  - Shared with full code and tasks for heat control, clock/calendar, day/night, soft light control, «cloud on the sky», display, buttons, I2C + SPI libraries and a radio client (`main` shown here)
- There probably is no electric motor inside the xCore machines. But to me it sounds like one!

Austrått manor

Austrått manor

Thank you!

This and more at http://www.teigfam.net/oyvind/home/technology/175-cpa-2018-fringe/

2 extra summary pages?

Task type	Usage
Normal	Tasks run on a logical core and run independently to other tasks. The tasks have predictable running time and can respond very efficiently to external events.
Combinable	Combinable tasks can be combined to have several tasks running on the same logical core. The core swaps context based on cooperative multitasking between the tasks driven by the compiler.
Distributable	Distributable tasks can run over several cores, running when required by the tasks connected to them.

From the XMOS Programming guide [1]

COLLECTED MOSTLY ON XCORE EXCHANGE AND FROM HERE

- Channels cannot be used between combined tasks
- A distributable task can be implemented very efficiently if all the tasks it connects to are on the same tile
- Distributed doesn't require its own cycles. Basically calling a function where the calling thread uses its cycles to run the code
- Either of these cause the compiler to emit different versions of the same task which may be
- Combined: Suitable for select combining, forked and run as an stand alone task
- Distributed: select flattening, inlined as a function
- In each case the functionality/logic is the same, however the implementation (and timing) may be significantly different
- If they are 'distributed' or 'combined' then the code runs in quite different ways viz
- Combined: on a another core co-operatively multitasking with other tasks
- Distributed: on the caller's stack/core
- Implicit 'decay' to weaker attributes viz: distributed->combined->single-core
- Indeed the compiler emits all the code necessary for the linker to chose the eventual usage - throwing the rest away!
- Thank you!