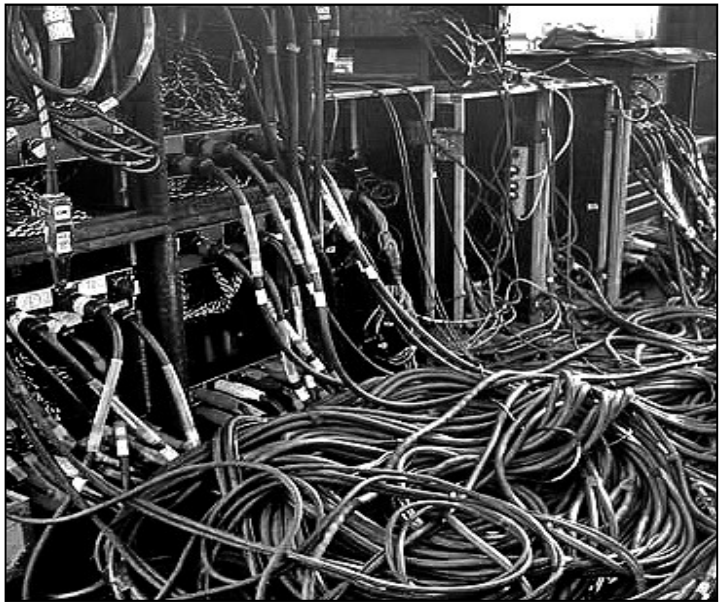
Department of Computer Science
University of Bristol

COMS20001 - Concurrent Computing

www.ole.bris.ac.uk/bbcswebdav/courses/COMS20001_2018/content



Lecture 04

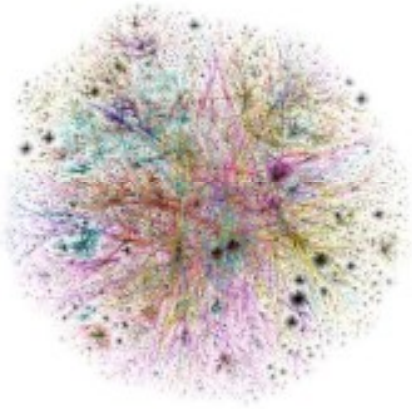
Inter-Process Data Exchange in xC

Sion Hannuna | hannuna@cs.bris.ac.uk
Tilo Burghardt | tilo@cs.bris.ac.uk
Dan Page | daniel.page@bristol.ac.uk

17 Slides

Recap: The Natural World is NOT serial ☺

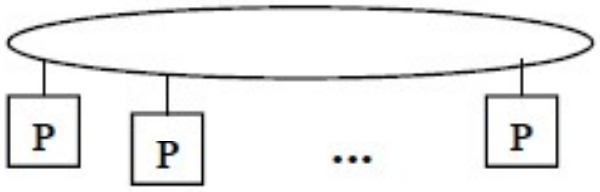
- …NATURE is massively concurrent !
  - natural networks tend to be continuously evolving, yet they are robust, efficient and long-lived
  - Concurrency is one of nature's core design mechanisms – and one of ours!

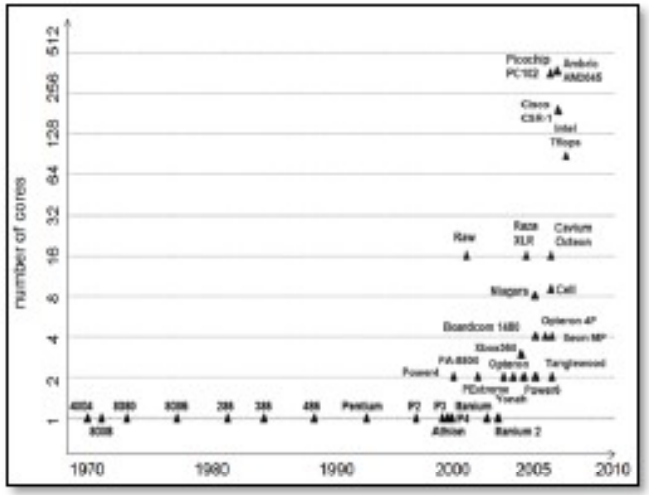


- in many cases **computing models phenomena of the real world**
  → computers are built as part of the physical world and can harvest natural concurrency for their own performance
  → concurrency can often help simplifying the modelling of systems

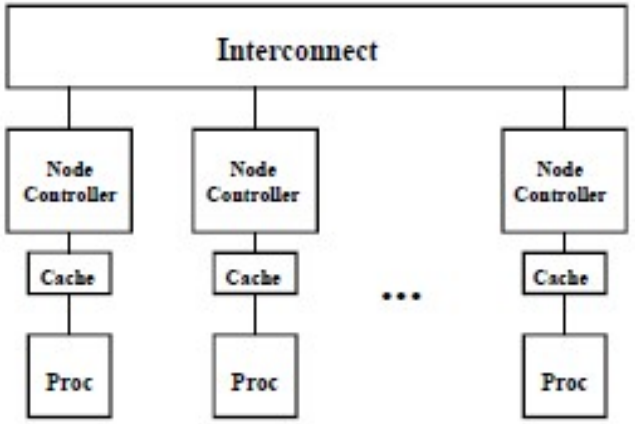
Recap: Multi-Processors and Multicore Revolution

- **Multiprocessors**
  (collection of communicating processors)
  - speed advantage by physically parallelised computation

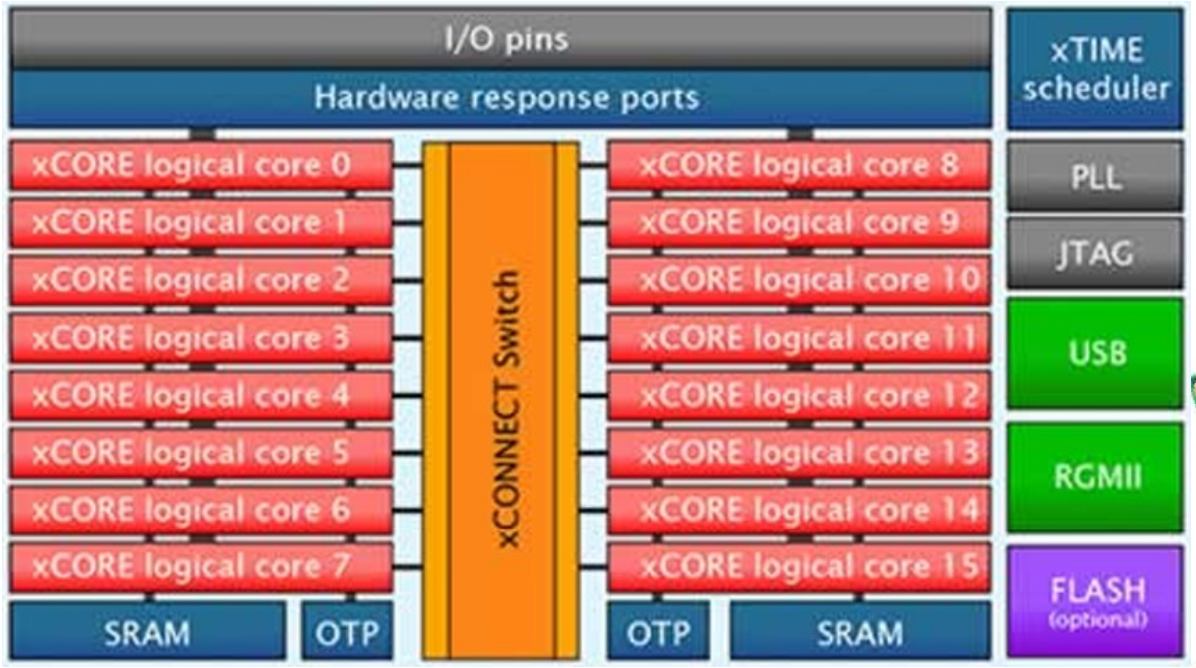


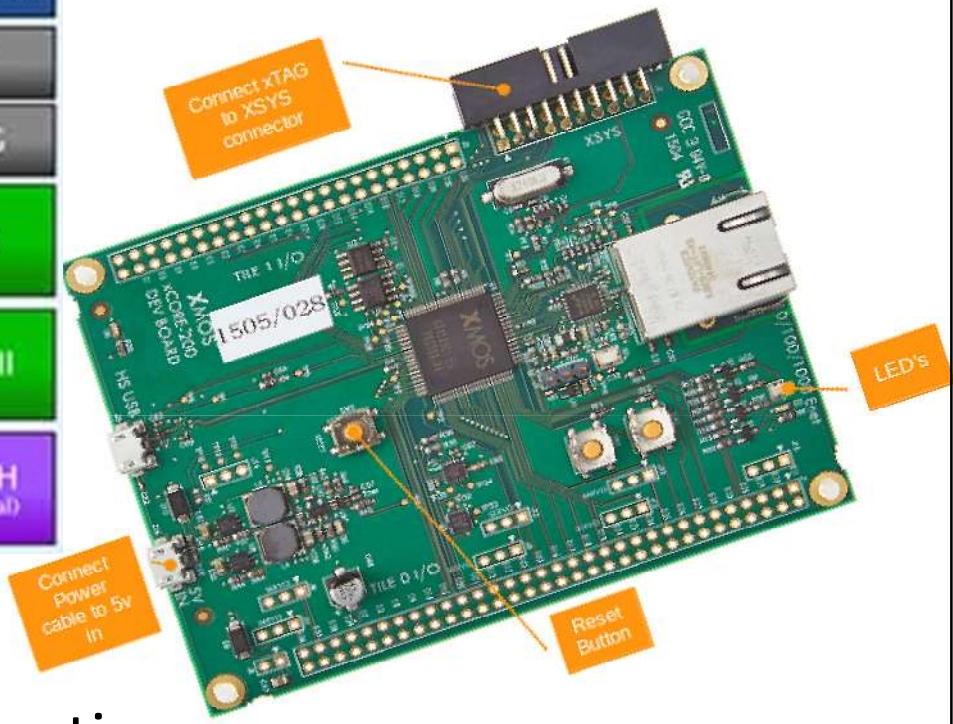


- **Multi-Memory Systems**
  - local, CPU-associated memory essential regardless of programming model
  - however, connectivity model affects specific performance tradeoffs



Recap: XMOS xCore200 Explorer Kit





- 16 logical cores on 2 xCORE tiles
- 32 channels for cross-core communication
- 512KB internal single-cycle SRAM (max 256KB per tile)
- 6 servo interfaces, 3D accelerometer, Gigabit Ethernet interface, 3-axis gyroscope, USB interface, xTAG debug adaptor, …

Recap: Sending and Receiving via a Channel

sendreceive.xc

```
#include <stdio.h>
#include <platform.h>

void receive ( chanend dataIncoming ) {
  char data;
  while (1) {
    dataIncoming :> data;
    printf("Received %i\n", data);
 } }

void send ( char data, chanend dataOutgoing ) {
while (1) {
   dataOutgoing <: data;
   printf ("Sent %i\n", data);
   data++ ;
} }

int main ( void ) {
  chan c;
  par {
    on tile[0] : receive (c);     // Thread 1
    on tile[1] : send(1, c);      // Thread 2
  }
  return 0;
}
```

...**wait here** until a data item is available on the channel...

...**wait here** until data has been delivered to the other end of the channel...

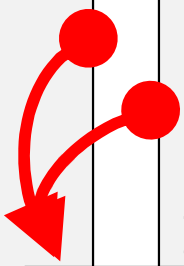
Main program

Recap: Interfaces for Single Client-Server Setups

interface.xc

```
//interface.xc
#include <platform.h>
#include <stdio.h>

//define a communication interface i
typedef interface i {
  void f(int x);
  void g();
} i;

//server task providing functionality of i
void myServer(server i myInterface) {
  int serving = 1;
  while (serving)
    select {
      case myInterface.f(int x):
        printf("f got data: %d \n", x);
        break;
      case myInterface.g():
        printf("g was called\n");
        serving = 0;
        break;
} } ...
```

```
...

//client task calling function
//of task 2
void myClient(client i myInterface) {
  myInterface.f(2);
  myInterface.f(1);
  myInterface.g();
}

//main starting two threads
//calling over an interface
int main() {
  interface i myInterface;
  par {
    myServer(myInterface);//only1server
    myClient(myInterface);//only1client
  }
  return 0;
}
```



Console | Problems | Task Viewer

```
<terminated> test1.xe [xCORE Application] xrun
f got data: 2
f got data: 1
g was called
```

Example: Multiple Interfaces – One Server `select`

interface2.xc

```
//interface2.xc
#include <platform.h>
#include <stdio.h>

//define a communication interface i
typedef interface i {
  void f(int x);
  void g();
} i;

//server task providing functionality of two IFs
void myServer(server i myInterface,
              server i myInterface2) {
  int serving = 1;
  while (serving)
    select { //SINGLE select statement for two IFs!!!
      case myInterface2.f(int x):
        printf("f got data in IF2: %d \n", x);
        break;
      case myInterface2.g():
        printf("g was called in IF2\n");
        break;
      case myInterface.f(int x):
        printf("f got data in IF1: %d \n", x);
        break;
      case myInterface.g():
        printf("g was called in IF1\n");
        serving = 0;
        break;
  }
} ...
```

```
...
//client task calling function of task 2
void myClient(client i myInterface,
              client i myInterface2) {
  myInterface.f(2);
  myInterface.f(1);
  myInterface2.f(3);
  myInterface2.f(4);
  myInterface2.g();
  myInterface.g();
}

//main starting two threads calling
//over two interfaces
int main() {
  interface i myInterface;
  interface i myInterface2;
  par {
    on tile[0]: myServer(myInterface,
                         myInterface2);
    on tile[1]: myClient(myInterface,
                         myInterface2);
  }
  return 0;
}
```

```
f got data in IF1: 2
f got data in IF1: 1
f got data in IF2: 3
f got data in IF2: 4
g was called in IF2
g was called in IF1
```

Example: Spot the Deadlock!

deadlockpossible.xc

```
//deadlockpossible.xc
#include <platform.h>
#include <stdio.h>

//define a communication interface i
typedef interface i {
  void f(int x);
  void g();
} i;

//server task providing functionality of i
void myServer(server i myInterface,
              server i myInterface2) {
  int serving = 1;
  while (serving)
    select {
      case myInterface2.f(int x):
        printf("f got data in IF2: %d \n", x);
        break;
      case myInterface2.g():
        printf("g was called in IF2\n");
        break;
      case myInterface.f(int x):
        printf("f got data in IF1: %d \n", x);
        break;
      case myInterface.g():
        printf("g was called in IF1\n");
        serving = 0;
        break;
  }
} ...
```

```
...
//client task calling functions
void myClient(client i myInterface) {
  myInterface.f(2);
  myInterface.f(1);
  myInterface.g();
}

//main starting three threads calling
//over two interfaces
int main() {
  interface i myInterface;
  interface i myInterface2;
  par {
    on tile[0]: myServer(myInterface,
                         myInterface2);
    on tile[1]: myClient(myInterface2);
    on tile[1]: myClient(myInterface);
  }
  return 0;
}
```

DEADLOCK POSSIBLE!

Example: Mixed Use of Channels and Interfaces

mix.xc

```
//mix.xc
#include <platform.h>
#include <stdio.h>

//define a communication interface i
typedef interface i {
  void f(int x);
  void g();
} i;

//server providing interface and channelend
void myServer(server i myInterface,
              chanend c) {
  int serving = 1;
  int data;
  while (serving)
    select {
      case c :> data:
        printf("Channel has %d \n", data);
        c <: data;
        break;
      case myInterface.f(int x):
        printf("f got data: %d \n", x);
        break;
      case myInterface.g():
        printf("g was called\n");
        serving = 0;
        break;
  }
} ...
```

```
 ...
//client task calling functions + doing channel com
void myClient(client i myInterface, chanend c) {
  int value;
  myInterface.f(1);
  c <: 5;
  c :> value;
  printf("Channel returned %d \n", value);
  myInterface.g();
}

//main starting two threads calling over interfaces
and channel
int main() {
  interface i myInterface;
  chan c;
  par {
    on tile[0]: myServer(myInterface,c);
    on tile[1]: myClient(myInterface,c);
  }
  return 0;
}
```

valuereturn.xc

```
//valuereturn.xc
#include <platform.h>
#include <stdio.h>

//define a communication interface i
typedef interface i {
  int f(int x);
  void g();
} i;

//server task providing functionality of i
void myServer(server i myInterface) {
  int serving = 1;
  while (serving)
    select {
      case myInterface.f(int x) -> int returnval:
        printf("f receives: %d \n", x);
        returnval = x * 2;
        break;
      case myInterface.g():
        printf("g was called\n");
        serving = 0;
        break;
} }

//client task calling functions
void myClient(client i myInterface) {
  printf("f returns: %d \n", myInterface.f(1));
  myInterface.g();
}

//main starting two threads
int main() {
  interface i myInterface;
  par {
    on tile[0]: myServer(myInterface);
    on tile[1]: myClient(myInterface);
  }
  return 0;
}
```

dataexchange.xc

```
//dataexchange.xc
#include <platform.h>
#include <stdio.h>

//define a communication interface i
typedef interface i {
  int f(int a[]);
  void g();
} i;

//server task providing functionality of i
void myServer(server i myInterface) {
  int serving = 1;
  while (serving)
    select {
      case myInterface.f(int a[]) -> int returnval:
        printf("f receives: %d \n", a[0],a[1]);
        a[1] = a[0]*2;
        returnval = a[0];
        break;
      case myInterface.g():
        printf("g was called\n");
        serving = 0;
        break;
} }

//client task calling functions
void myClient(client i myInterface) {
  int a[2] = {1,0};
  printf("f returns: %d \n", myInterface.f(a));
  printf("a[1] set to: %d \n", a[1]);
  myInterface.g();
}

//main starting two threads
int main() {
  interface i myInterface;
  par {
    on tile[0]: myServer(myInterface);
    on tile[1]: myClient(myInterface);
  }
  return 0;
}
```

memcopy.xc

```
//memcopy.xc
#include <platform.h>
#include <stdio.h>
#include <string.h>

//define a communication interface i
typedef interface i {
  int f(int a[]);
  void g();
} i;

//server task providing functionality of i
void myServer(server i myInterface) {
  int serving = 1;
  int data[2] = {10,11};
  while (serving)
    select {
      case myInterface.f(int a[]) -> int returnval:
        printf("setting buffer \n");
        memcpy(a, data , 2*sizeof(int));
        returnval = a[0];
        break;
      case myInterface.g():
        printf("g was called\n");
        serving = 0;
        break;
} }

//client task calling functions
void myClient(client i myInterface) {
  int a[2] = {0,0};
  printf("f returns: %d \n", myInterface.f(a));
  printf("a set to: [%d,%d] \n", a[0], a[1]);
  myInterface.g();
}

//main starting two threads
int main() {
  interface i myInterface;
  par {
    on tile[0]: myServer(myInterface);
    on tile[1]: myClient(myInterface);
  }
  return 0;
}
```

interfacearrrays.xc

```
//interfacearrays.xc
#include <platform.h>
#include <stdio.h>
#include <string.h>

//define a communication interface i
typedef interface i {
  int f(int a[]);
  void g();
} i;

//server task providing functionality of i
void myServer(server i myInterface[n], unsigned n) {
  int serving = 1;
  int data[2] = {10,11};
  while (serving)
    select {
      case myInterface[int j].f(int a[]) -> int returnval:
        printf("f called from %d \n",j);
        memcpy(a, data , 2*sizeof(int));
        returnval = a[0];
        break;
      case myInterface[int j].g():
        printf("g was called from %d\n",j);
        serving = 0;
        break;
} }

//client task calling functions
void myClient(client i myInterface, int j) {
  int a[2] = {0,0};
  printf("f returns: %d \n", myInterface.f(a));
  printf("a set to: [%d,%d] \n", a[0], a[1]);
  if (j==1) myInterface.g();
}

//main starting two threads
int main() {
  interface i myInterface[2];
  par {
    on tile[0]: myServer(myInterface,2);
    on tile[1]: {
                  myClient(myInterface[0],0);
                  myClient(myInterface[1],1);
                }
  }
  return 0;
}
```

channelarrays.xc

```
//channelarrays.xc
#include <platform.h>
#include <stdio.h>

void taskA(chanend c[n], unsigned n) {
  int serving = 1;
  while (serving)
    select {
      case c[int j] :> int data:
        printf("channel %d receives %d\n",j,data);
        c[j] <: data;
        if (data == 0) serving = 0;
        break;
} }

void taskB(chanend c, chanend d, int terminate) {
  int data;
  c <: 1;
  c :> data;
  c <: 2;
  c :> data;
  if (terminate == 1) {
      d :> data;
      c <: 0;
      c :> data;
  } else d <: 0;
}

int main() {
  chan c[2],d;
  par {
    on tile[0]: taskA(c,2);
    on tile[1]: taskB(c[0],d,1);
    on tile[1]: taskB(c[1],d,0);
  }
  return 0;
}
```

deadlock2.xc

```
//deadlock2.xc
#include <platform.h>
#include <stdio.h>

void taskA(chanend c[n], unsigned n) {
  int serving = 1;
  while (serving)
    select {
      case c[int j] :> int data:
        printf("channel %d gets %d\n",j,data);
        c[j] <: data;
        if (data == 0) serving = 0;
        break;
} }

void taskB(chanend c, chanend d, int terminate) {
  int data;
  c <: 1;
  c :> data;
  c <: 2;
  c :> data;
  if (terminate == 1) {
      d :> data;
      c <: 0;
      c :> data;
  } else d <: 0;
}

int main() {
  chan c[4],d,e;
  par {
    on tile[0]: taskA(c,4);
    on tile[0]: taskB(c[0],e,1);
    on tile[0]: taskB(c[1],e,0);
    on tile[1]: taskB(c[2],d,1);
    on tile[1]: taskB(c[3],d,0);
  }
  return 0;
}
```

SPOT THE DEADLOCK !!!

shutdowncounter.xc

```
//shutdowncounter.xc
#include <platform.h>
#include <stdio.h>

void taskA(chanend c[n], unsigned n) {
  int serving = 1;
  int counter = 0;
  while (serving)
    select {
      case c[int j] :> int data:
        printf("channel %d receives %d\n",j,data);
        c[j] <: data;
        if (data == 0) counter++;
        if (counter == 4) serving = 0;
        break;
} }

void taskB(chanend c) {
  int data;
  c <: 1;
  c :> data;
  c <: 0;
  c :> data;
}

int main() {
  chan c[4];
  par {
    on tile[0]: taskA(c,4);
    on tile[0]: taskB(c[0]);
    on tile[0]: taskB(c[1]);
    on tile[1]: taskB(c[2]);
    on tile[1]: taskB(c[3]);
  }
  return 0;
}
```

nestedpar.xc

```
//nestedpar.xc
#include <platform.h>
#include <stdio.h>

void taskA(chanend c[n], unsigned n) {
  int serving = 1;
  int counter = 0;
  while (serving)
    select {
      case c[int j] :> int data:
        printf("channel %d gets %d\n",j,data);
        c[j] <: data;
        if (data == 0) counter++;
        if (counter == n) serving = 0;
        break;
} }

void taskB(chanend c, chanend d) {
  int data1, data2;
  par {
    { d <: 1; d :> data1; }
    { c <: 2; c :> data2; }
  }
  par {
    { d <: 0; d :> data1; }
    { c <: 0; c :> data2; }
  }
}

int main() {
  chan c[4];
  par {
    on tile[0]: taskA(c,4);
    on tile[0]: taskB(c[0],c[1]);
    on tile[1]: taskB(c[2],c[3]);
  }
  return 0;
}
```

Some Common Errors Part I

```
#include <platform.h>
#include <stdio.h>

int a[2] = {1,2};

void taskA(chanend c, int a[]) {
  c <: a[0];
}

void taskB(chanend c, int a[]) {
  c :> a[1];
}

int main() {
  chan c;
  par {
    on tile[0]: taskA(c,a);
    on tile[1]: taskB(c,a);
  }
  return 0;
}
```

DOES NOT COMPILE

```
#include <platform.h>
#include <stdio.h>

int a = 2;

void taskA(chanend c, int a) {
  c <: a;
}

void taskB(chanend c) {
  c :> int i;
}

int main() {
  chan c;
  par {
    on tile[0]: taskA(c,a);
    on tile[1]: taskB(c);
    on tile[1]: taskB(c);
  }
  return 0;
}
```

DOES NOT COMPILE

Some Common Errors Part II

```
#include <platform.h>
#include <stdio.h>

int a = 2;

void taskA(chanend c) {
  c <: 2;
  a = 2;
}

void taskB(chanend c) {
  int i;
  c :> i;
  a = i;
}

int main() {
  chan c;
  par {
    on tile[0]: taskA(c);
    on tile[1]: taskB(c);
  }
  return 0;
}
```

DOES NOT COMPILE

```
#include <platform.h>
#include <stdio.h>

void taskA(chanend c) {
  c <: 2;
}

void taskB(chanend c) {
  c :> int i;
}

int main() {
  chan c;
  par {
    on tile[0]: taskA(c);
    on tile[1]: taskB(c);
  }
  par {
    on tile[0]: taskA(c);
    on tile[1]: taskB(c);
  }
  return 0;
}
```

DOES NOT COMPILE

Some Common Errors Part III

```
#include <platform.h>
#include <stdio.h>

void taskA(chanend c) {
  c <: 2;
}

void taskB(chanend c) {
  chan d;
  par {
      on tile[0]: taskA(d);
      on tile[1]: d :> int j;
  }
  c :> int i;

}

int main() {
  chan c;
  par {
    on tile[0]: taskA(c);
    on tile[1]: taskB(c);
  }
  return 0;
}
```

DOES NOT COMPILE

```
#include <platform.h>
#include <stdio.h>

void taskA(chanend c) {
  c <: 2;
}

void taskB(chanend c) {
  chan d;
  par {
      taskB(d);
      d :> int j;
  }
  c :> int i;
}

int main() {
  chan c;
  par {
    on tile[0]: taskA(c);
    on tile[1]: taskB(c);
  }
  return 0;
}
```

RUN TIME ERROR: ILLEGAL RESOURCE

Looking ahead...

Replication and Pipelining