Lucien Murray-Pitts
Mentor Graphics,
Front End Solutions Group

Presentation to Company Name

By Celoxica Presenter

Whats MoC?

- **Model of Computation**
  - **A set of operations that can be used describe and evaluate a computational system.**

- **I prefer to now think of it as Model-On-Chip**

- **Promised Opportunities**
  - **Algorithm Acceleration**
    - **Exploit parallelism to increase performance with custom HW implementation**
  - **Algorithm Offload**
    - **Free CPU resource by offloading bottleneck processes**

- **BIG Challenges – Some unique to MoC Computing…. some not!**
  - **Development complexity**
    - **Design framework and methods, deployment and integration/middleware**
  - **Coupling to coprocessor/Data bandwidth**
  - **Application specific**
  - **Price/Performance/Power!**
  - **Choosing the right applications!**

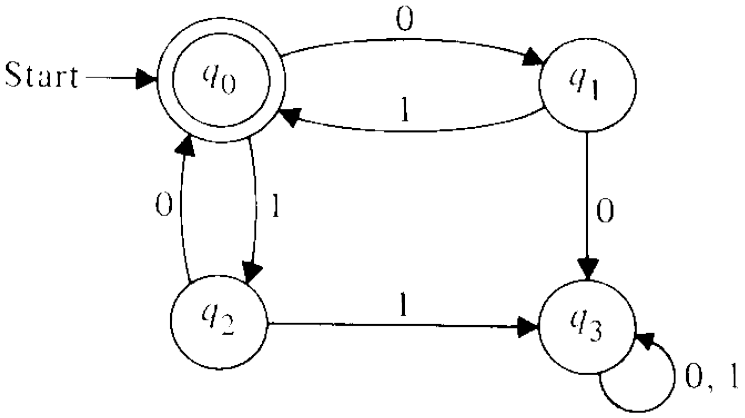
Performance Comparisons

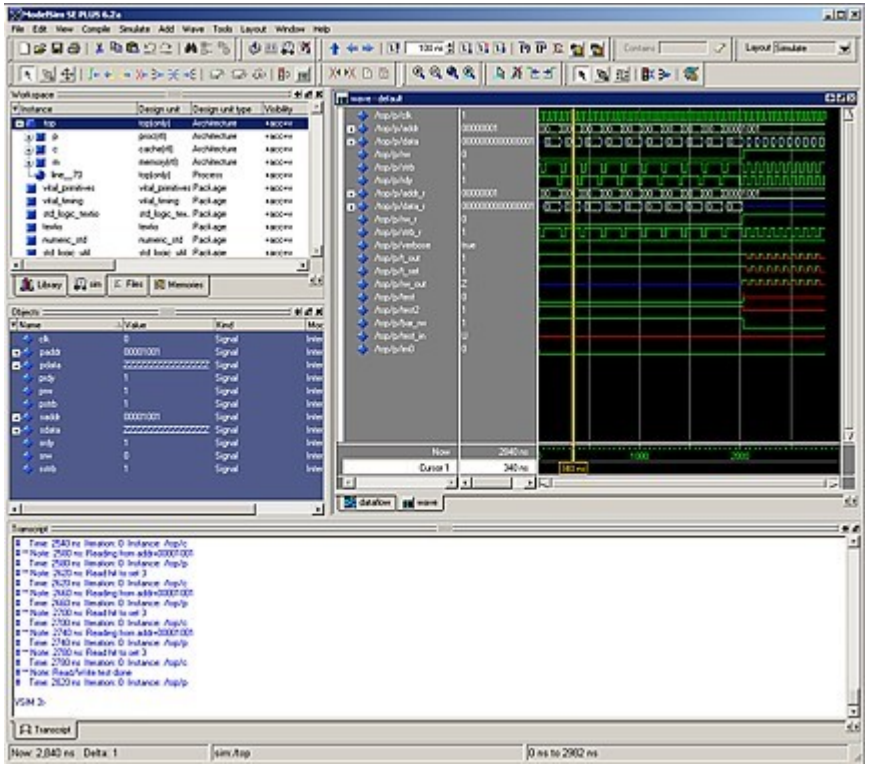
	Microprocessor (P4)	FPGA(2VP100)
Clock Speed	3.8GHz	180MHz
Internal Memory Bandwidth	122 GBytes per Sec	7.5 TBytes per Sec
# Floating Point Units	2	146
Power Consumption	>100 WATTS	<10 WATTS
Peak Performance	7.6 GFLOPs	26 GFLOPS
Sustained Performance	0.76 GFLOPs	13 GFLOPS
I/O / External Memory Bandwidth	8.5 GBytes/sec	67 GBytes/sec

Source: AC –

How are we doing it today?

- **Languages**
  - VHDL, VerilogPrecisely timed, synchronization discreetly controlled
- **Debugging**
  - Wave form, State Very detailed description, obscures intent





What else could we use?

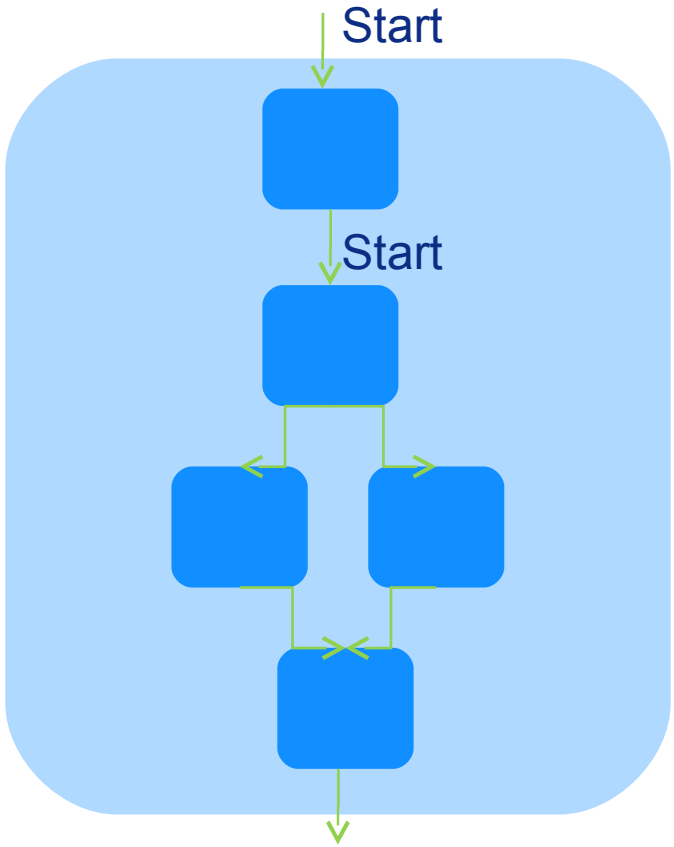
- State;
  - FSA, Petri-Nets
- Process
  - Process Algebra; CSP
- Stream;
  - KPN, Dataflow Nets
- And more recently SDF

- An initial model of a system may start at a KPN level, being revised down into to a structural model of HW using TLM bus communication and the SW for DSP using SDF.

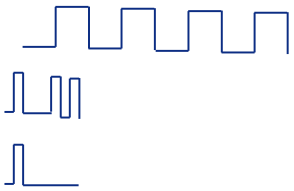
But MoC are too complex!

- **CSP Defines**

CSP		occam
STOP	primitive process	STOP
SKIP	primitive process	SKIP
$e \rightarrow P$	event prefix	CHAN I/O, *see text*
$P;Q$	sequence	SEQ
$P \|\| Q$	interface parallel	PAR, *see text*
$P \|\|\| Q$	interleaving parallel	PAR, *see text*
$P \sqcap Q$	nondeterministic choice	not needed
$P \Box Q$	deterministic choice	ALT
$P\ e$	hiding	local CHANs
$f(P)$	relabelling	translator PROCs
name	process reference	PROC call
$\mu\ name \bullet P$	recursion	transform into iteration



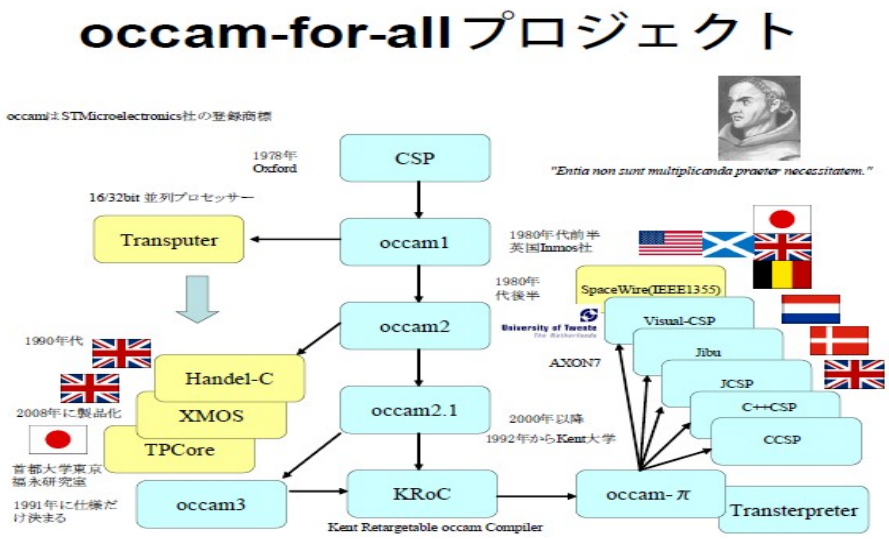
It's a new beginning…



Good bye wave simulation

Change in level of abstraction

- OCCAM inspired the concept for abstract Hardware design
  - Oxford Hardware Compilation Group
- HandelC, BachC, and early CyberC born from this branch of research
  - Fixed timing behaviour but flexible inter-stage timing (C based)
- A decade later SpecC, StreamsC, … and finally SystemC came to dominate
  - Flexible timing intra/inter-stage (C to C++ based)



- C subset, extended with OCCAM keywords to allow easy modelling
  - Handel-C uses multiple main() functions to support multiple clock domains
  - Input clock can be:
    - Same
    - Same, divided differently
    - Different (Asynchronous)

```
// Domain1.c
set clock = external_divide "P13" 4;
void main (void)
{
    …
}

// Domain2.c
set clock = external "P13";
void main (void)
{
    …
}
```

Basic CHAN concept

- Must be uni-directional point-to-point
- First use defines their direction and the domains in which they transmit and receive.
- Channels used between clock domains must be defined in one file and then declared as extern in another.
- The timing between domains is unspecified, but the transmission is guaranteed to occur

```
// Domain1.hcc
set clock = external “CK1";
chan sync;
void main (void)
{
   unsigned 1 pulseRx;
   …
   sync ? pulseRx;
}

// Domain2.hcc
set clock = external_divide “CK2“ 4;
extern chan sync;
void main (void)
{
   unsigned 1 pulseTx;
   …
   sync ! pulseTx;
}
```

Pipeline Example

- **Add logic to fill and clear the pipe**
- **For 1-cycle latency**
  - **Don't output first result**
  - **Input dummy values on last cycle to push the last result out**

```
void main( void )
{
   unsigned i;
   signal unsigned 6 j, k;
   i = 0;
   while( i <= elements )
   {
      par {
         if( i < elements )
                p_l_Adder(A_ROM[i], B_ROM[i], C_ROM[i], &j);
         else
                p_l_Adder(0, 0, 0, &j);
         if( i )
            par{
          output1 ! j;
          output2 ! i;
            }
         else  delay;
         i++;
      }
   }
}
```

Modeling Complex Mathematics

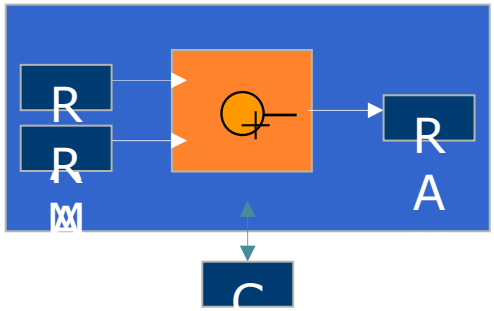
Turning Software into Silicon

HPC Streams

```
HPC_INT32 (ij);     /* Address(i, j) @ input */
HPC_FLOAT (Aij);    /* A(i, j)               */
HPC_FLOAT (Bij);    /* B(i, j)               */
HPC_FLOAT (Cij);    /* C(i, j) = A(i, j) + B(i, j)*/
HPC_VOID  (Done);   /* RAM write completed */
```

Main Program

```
void main()
{
          ...
          while(1) par
          {
   ...
       Calc();
       Control();
   ...
          }
}
```



Calculation Pipeline

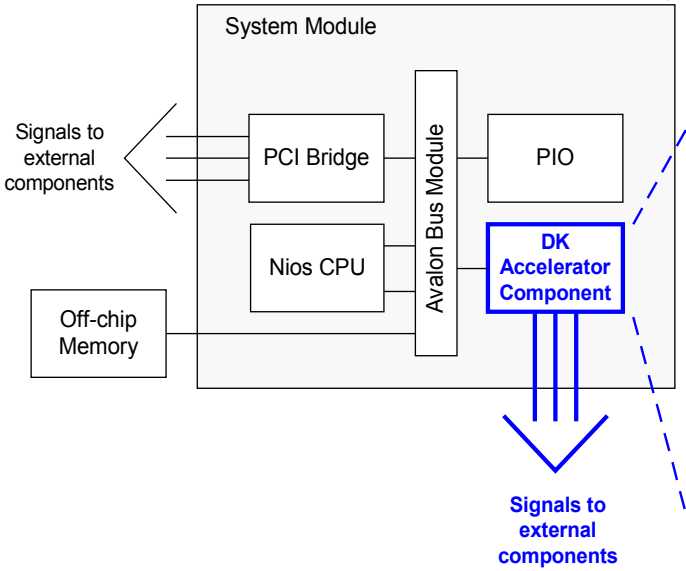
```
macro proc Calc()
{
  par{
   HpcInput        (&ij);
   HpcPL2RAMRead   (&ij,  &Aij, PL2RAMA);
   HpcPL2RAMRead   (&ij,  &Bij, PL2RAMB);
   HpcAdd          (&Aij, &Bij, &Cij);
   HpcPL2RAMWrite  (&ij,  &Cij, &Done, PL2RAMC);

   /* Monitor pipeline activity */
   HpcMonitor (&ij, &Done, &NetworkActive);
  }
}
```

Data Feed/Control

```
macro proc Control()
{
 do /* Loop over rows i*/ {
  do  /*Loop over columns j*/ {
          /* Issue computations for this stage*/
   ...
      HpcWrite (&ij, Addr);
   ...
  }
 }
          while (NetworkActive) delay;
          /* Output Results */
}
```

- **PixelStreams live video filter network**
  - **Video processing, Video compositing**

- **CPU (Altera Nios II) embedded processor**
  - **User interface via touch-screen, Configuration of filter networks**




```
/*
 *  Avalon bus macros.
 *  Handle reads and writes to/from the component.
 */
macro proc WriteData(Offset, WriteData, ByteEnable, Ready)
{
        // 32-bit data --
        //   DTheta (11)
        //   DPhi   (11)
        //   VX     ( 4)
        //   VY     ( 5)
        //    R     ( 1)
        //    G     ( 1)
        //    B     ( 1)
        Sphere[Offset <- 2].DTheta = WriteData[21:11];
        Sphere[Offset <- 2].VX = WriteData[10:7];
        Sphere[Offset <- 2].VY = WriteData[ 6:3];
        Sphere[Offset <- 2].Red  = WriteData[ 2];
        Sphere[Offset <- 2].Green  = WriteData[ 1];
        Sphere[Offset <- 2].Blue  = WriteData[ 0];

        Ready = 1;
}

macro proc ReadData(Offset, ReadData, Ready)
{
        ReadData = 0
         @Sphere[Offset<-2].DTheta
         @Sphere[Offset<-2].VX
         @Sphere[Offset<-2].VY
         @Sphere[Offset<-2].Red
         @Sphere[Offset<-2].Green
         @Sphere[Offset<-2].Blue;
        Ready = 1;
}
```

Simple Counter Slave

Custom Peripheral Bus Interface
enables processor communications via the CPU Bus

Component Declaration
Declare the Bus slave

Custom C Algorithm
Design custom component for product differentiation

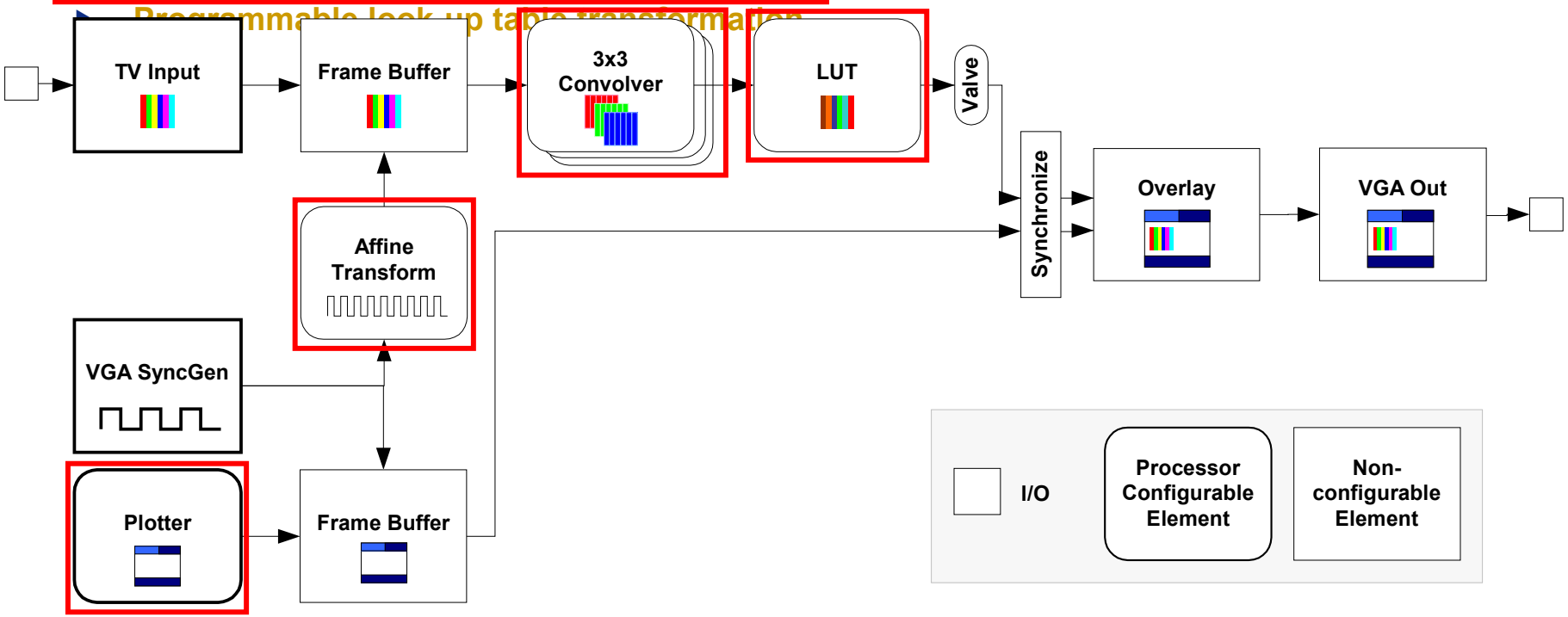
```
unsigned 32 Counter;
unsigned  1 Go;

macro proc WriteData(Offset, WriteData, ByteEnable, Ready)
{
    Go = WriteData[0];  // start or stop, depending on the LSB
    Ready = 1; // signal completion
}

macro proc ReadData(Offset, ReadData, Ready)
{
    ReadData = Counter; // read the counter value
    Ready = 1;  // signal completion
}
```

```
void main()
{
    par
    {
        DeclareSOPCSlave(WriteData, ReadData); // simple slave
        RunCounter();   // main application code
    }
}
```

```
macro proc RunCounter()
{
    while(1)
    {
        if(Go)
            Counter++;
    }
}
```

Stream based Video Filter Network

- Plotter for user interface display
- Programmable affine transformation
- Programmable convolver
- Programmable look-up table transformation

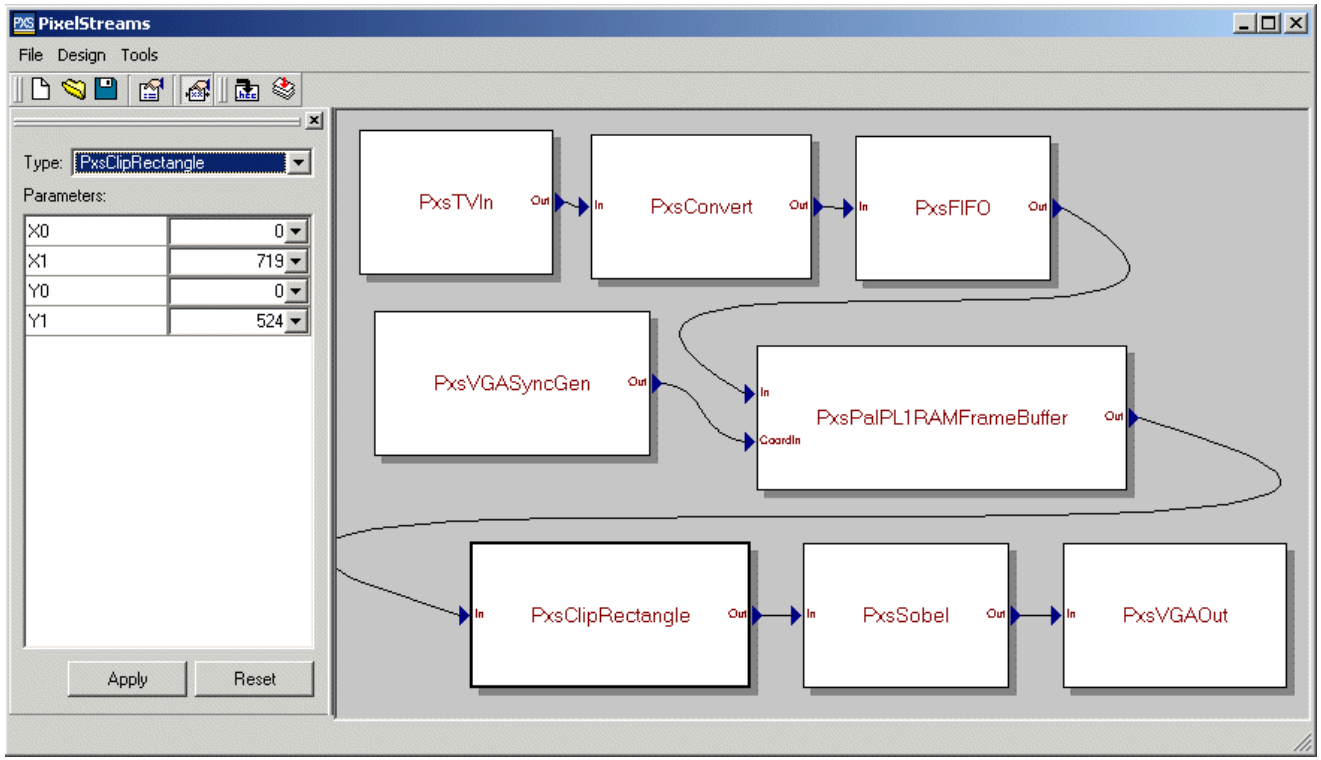


Celoxica/Agility Altera Based RC250 Board

- Large Altera Stratix II FPGA EP2S90 device
- Memory
  - 4banks x 1M x 36bit ZBT SSRAM (16Mbytes total)
  - 1 bank x 16M x 32bit SDRAM
- Video I/O
  - Composite video in
  - S-Video in/out
  - VGA out
  - DVI in/out
- AC97 compatible audio
- Dual Gigabit Ethernet
- USB interface
- SD card interface

RC250

- **Image Processing Block design, with Interconnecting Streams**

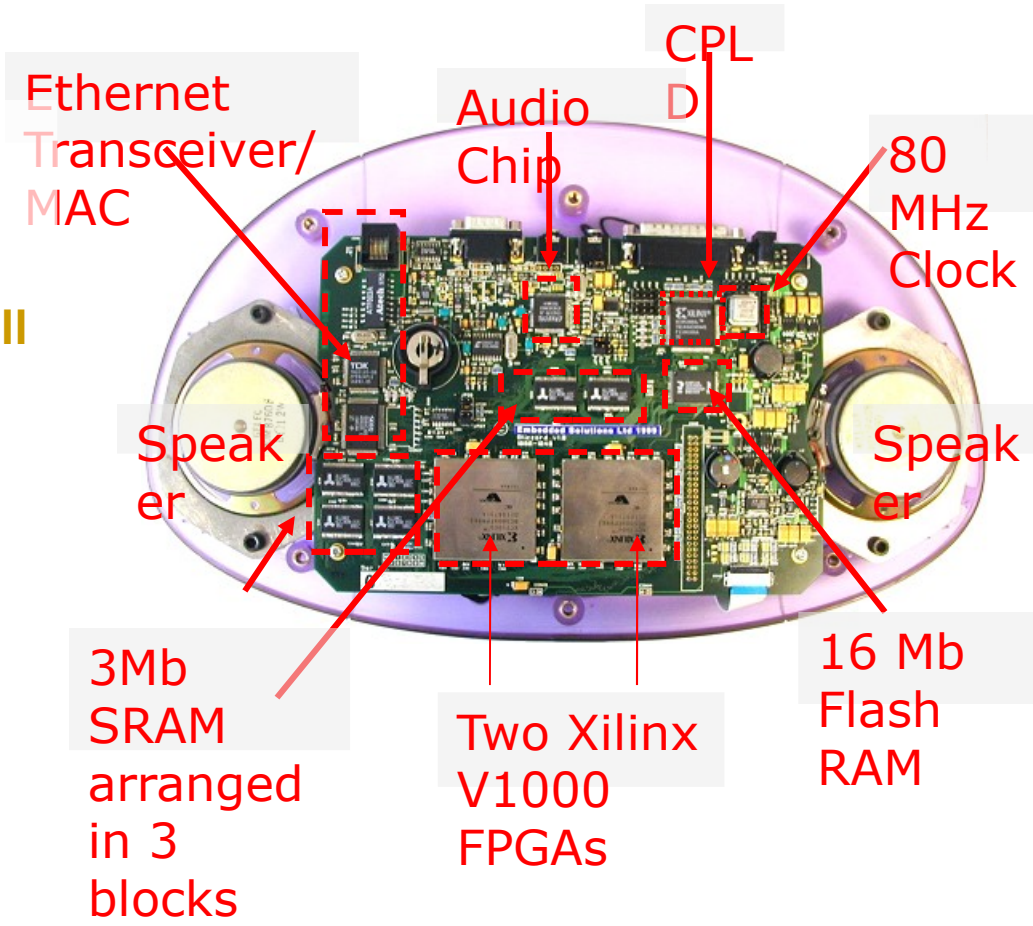


- **Program**
- **`YCbCrIn`, `GreyIn`, `FIFO`, `VGASync`, `Out`, `Sobel` are all specialized streams**

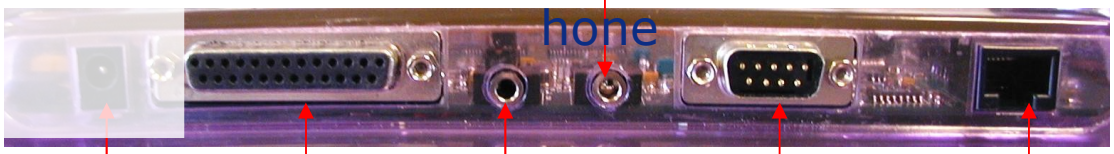
```
par
{
    PxsVideoIn            (&YCbCrIn, 0, 0, ClockRate);
    PxsConvert            (&YCbCrIn, &GreyIn);
    PxsFIFO               (&GreyIn, &FIFO, 256);
    PxsVGASyncGen         (&VGASync, SYNCGEN_MODE_1024_768_60HZ);
    PxsPL1RAMFrameBuffer  (&FIFO, &VGASync, &FB, 1024, PXS_BOB,
                           PalPL1RAMCT (0), ClockRate);
    PxsSobel              (&FB, &Sobel, 1024);
    PxsConvert            (&Sobel, &Out);
    PxsVideoOut           (&Out, 0, 0, ClockRate);
}
```

Photograph of Demo

- 4 Engineers
  - 1 hardware
  - 3 software
- 3 months for VoIP, Board and all demos!
- Interfacing to
  - ISA CystalWave Audio Chip
  - TI PCI Ethernet Chip
  - SRAM
  - Flash
  - Parallel Port
  - Inter FPGA communications



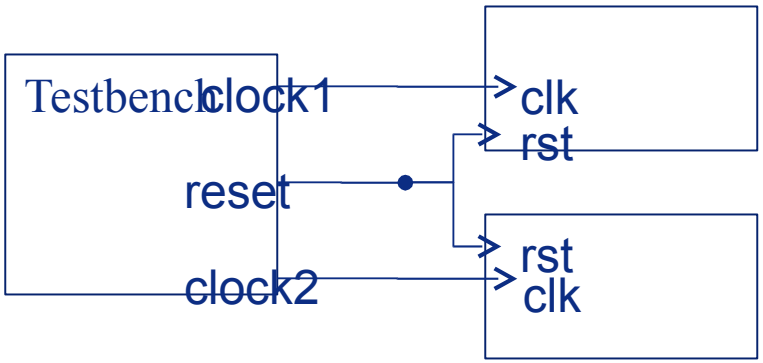
- **Design a MultiMedia platform with**
  - Software like functionality
  - FPGA based
  - Entirely using
- **VoIP Telephony**
  - 1) G711 codec (mLaw)
  - 2) H.323 "Faststart" protocol
    - Including H225 and Q931
  - TCP Control Connection
  - Sound over UDP
- **Reconfigurable from Web Server**
  - Tetris, Boat Game, Space Invaders
  - Screen Savers
  - MP3 player



SystemC for Modelling?

- Originally Intended for modelling, now encompasses complete complex systems
- Extension for Analogue underway
- As you can see its very hard to model all that abstractly because of modelling issues between parallel bocks.

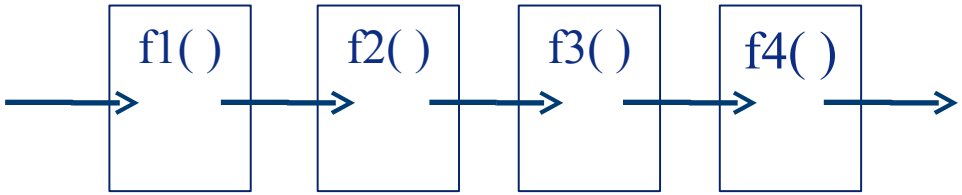
```
#include "systemc.h"

SC_MODULE(adder)           // module (class) declaration
{
  sc_in<int> a, b;         // ports
  sc_out<int> sum;

  void do_add()            // process
  {
    sum.write(a.read() + b.read()); //or just sum = a + b
  }

  SC_CTOR(adder)           // constructor
  {
    SC_METHOD(do_add);     // register do_add to kernel
    sensitive << a << b;   // sensitivity list of do_add
  }
};
```



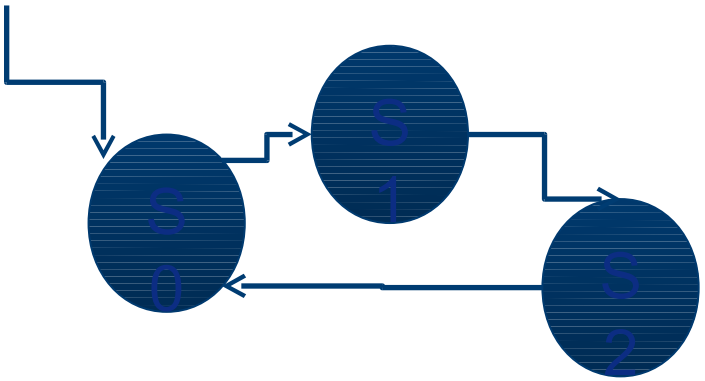

```
sc_main( )

{

  testbench tb ;

  top top_system1 ;

  addr top_system2 ;


  tb.clock1( top_system1.clk ) ;

  tb.reset( top_system1.rst ) ;


  tb.clock2( top_system2.clk ) ;

  tb.reset( top_system2.rst ) ;


  sc_start( … );
```

SystemC Modeling

```
SC_MODULE( mult ) {
    sc_clk_in       clk;
    sc_in<uint17>   a;
    sc_in<uint17>   b;
    sc_out<uint17> mult;
    uint35          mult;
    uint17          a_in, b_in;
    uint35          pipe_1 ;
    uint35          pipe_2, pipe_3;
    SC_CTOR( mult ) {
      SC_METHOD( pipe1, clk.pos());
  }
  #pragma iomode fixed
  void pipe2( ) {
   switch( S )
   {
      case S0:
        a_in = a ; b_in = b ;
        pipe_1=f1(a_in,b_in);
        S++; break;
      case S1:
        pipe_2=f2(pipe_1); S++; break;
      case S2:
        res=f3(pipe_2); S=S0; break;
  }
```

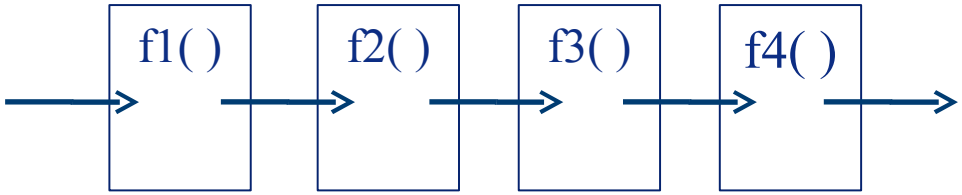


eg. function f1(a, b) = a*b ;

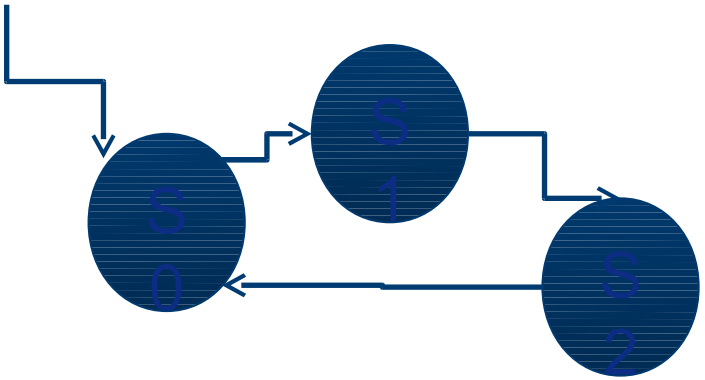


Looks very similar to another language?


```
module mult(clk, a, b, res);
input         clk;
input  [17:0] a;
input  [17:0] b;
output [35:0] mult;
reg    [35:0] mult;
reg    [17:0] a_in, b_in;
reg    [35:0] pipe_1 ;
reg    [35:0] pipe_2, pipe_3;
always @(posedge clk)
begin
   case( S )
      S0: begin
        a_in = a ; b_in = b ;
        pipe_1=f1(a_in,b_in);
        S=S1; end
      S1: begin
        pipe_2=f2(pipe_1); S=S2; end
      S2: begin
        res<=f3(pipe_2); S=S0; end
   endcase
end
```

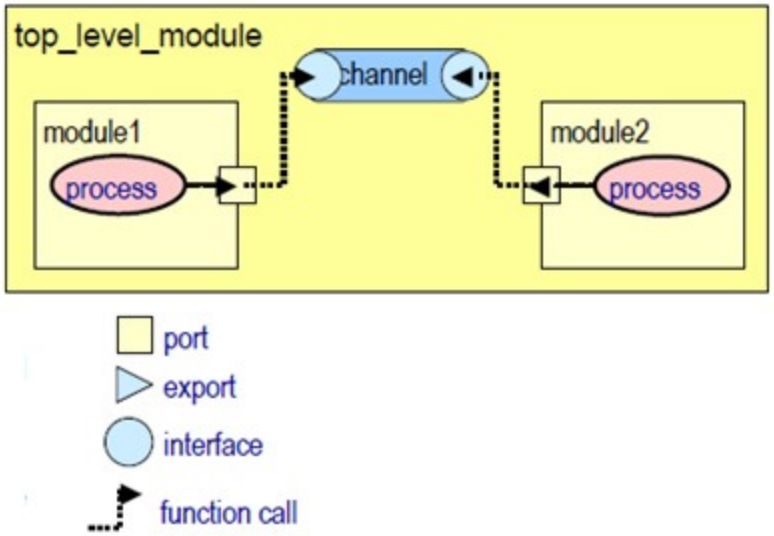


eg. function f1(a, b) = a*b ;



SystemC Ports (Slightly Better)

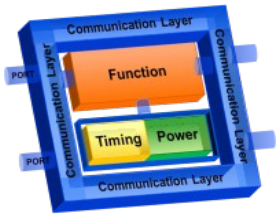
- **Exporting**
  - sc_port
  - sc_channel / sc_interface

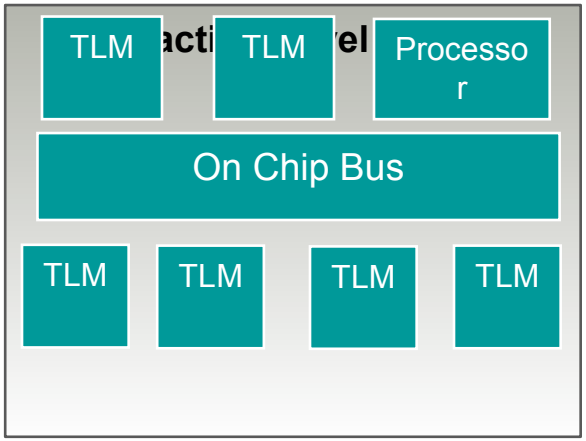


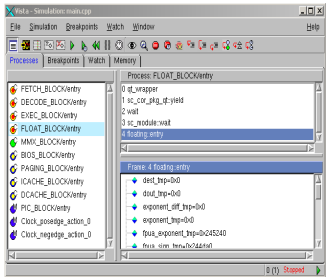

```
class write_if: virtual public sc_interface
{public: virtual bool write(char) = 0;
         virtual void reset() = 0;  };

class read_if: virtual public sc_interface
{public: virtual bool read(char&) = 0;  };

class io_handler : public sc_module,
  public write_if, public read_if
{
    bool write( char c ) { ... Code ... }
    bool read( char &c ) { ... Code ... }
    void reset( ) {}
};

SC_MODULE (INP) {
public:  sc_port<read_if> IPORT ;
  // ... code ...
};
SC_MODULE (OUTP) {
public:  sc_port<write_if> OPORT ;
  // ... code ...
};
SC_MODULE ( TEST_SYSTEM ) {
        sc_in<bool> clk ;

        OUTP Module1 ;
```

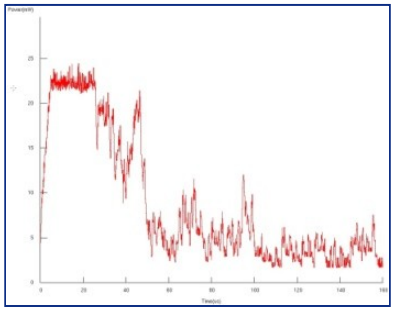
Recent Modeling (Visa TLM)

Modeling | **Assembly** | **Debug** | **Analysis**

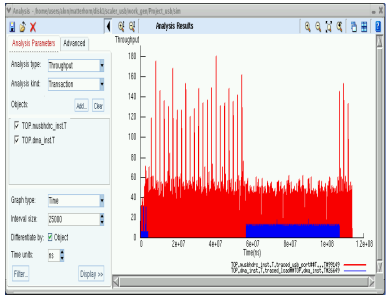






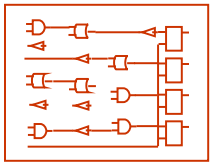


Power



Performance

(latency, utilization, bandwidth)



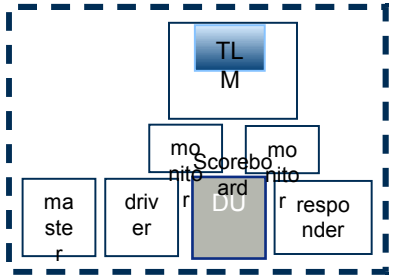
RTL IP

High Level Synthesis

RTL

Virtual Prototype

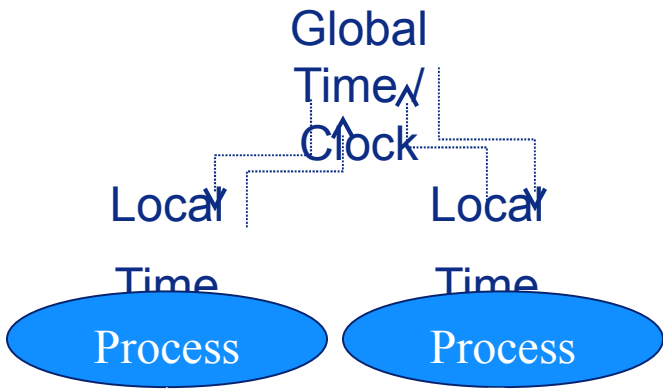
Software Development



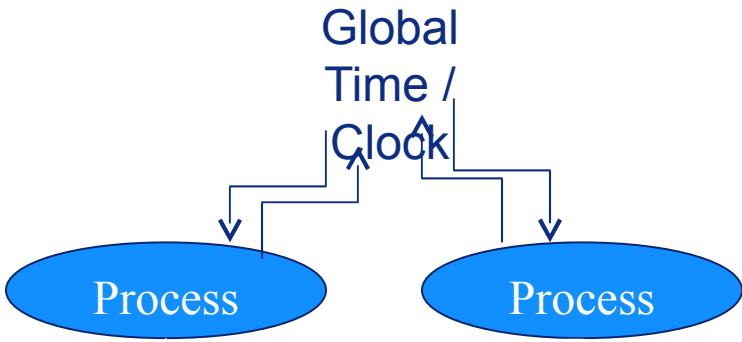
RTL Verification Using OVM

Transaction Level Modeling

- Loosely-timed = as fast as possible
  - Only sufficient timing detail to boot O/S and run multi-core systems
  - Processes can run ahead of simulation time (temporal decoupling)
  - Each transaction completes in one function call
  - Uses direct memory interface (DMI)



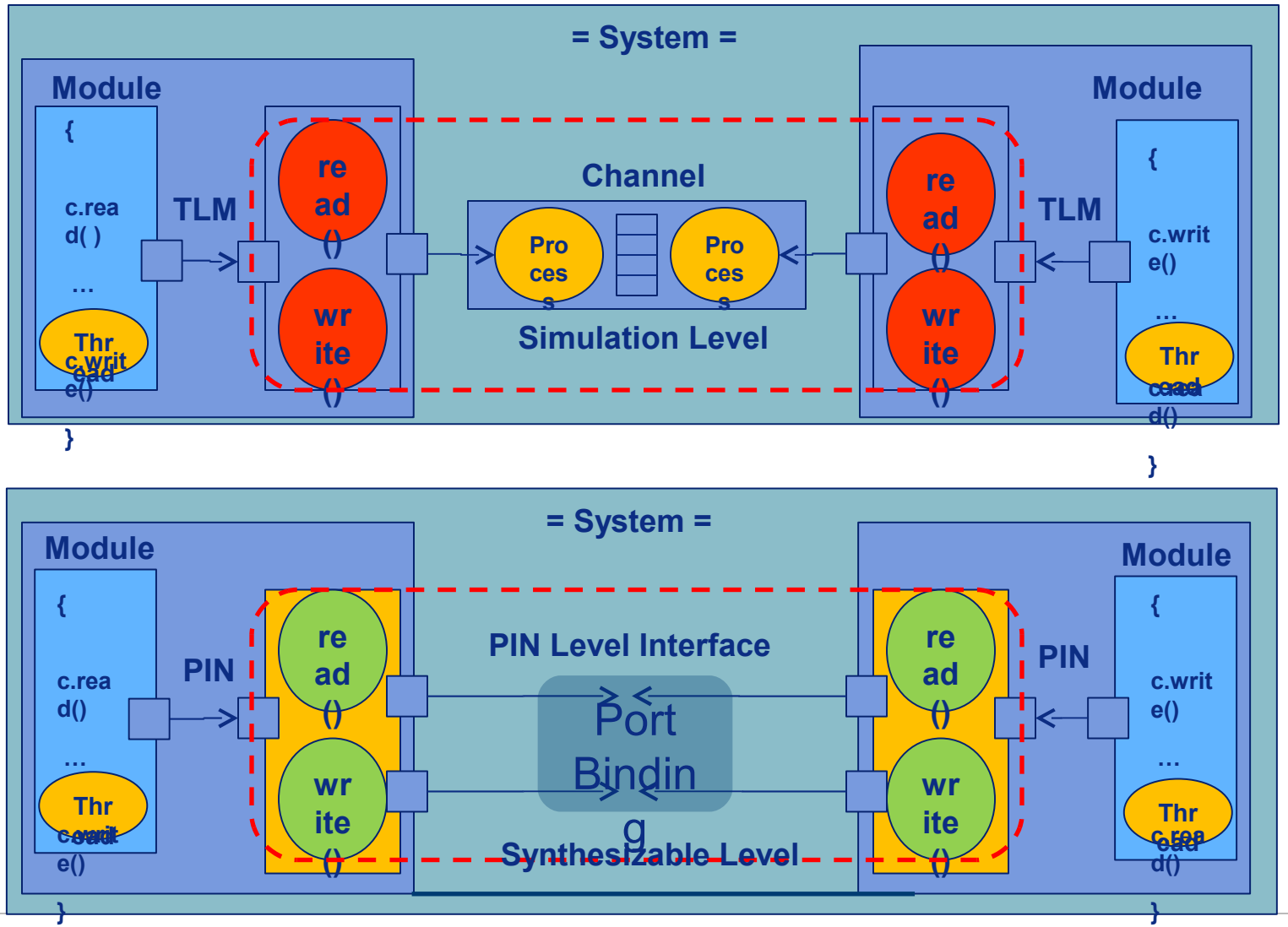
- Approximately-timed = just accurate enough for performance modeling
  - *aka* cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - Processes run in lock-step with simulation time
  - Each transaction has 4 timing points (extensible)



- Simulation Time improves
- This assists with lower number of late-stage glitches
- Fuller coverage then ever before
- Previous IP can be simulated at high level, concentrating on RTL verification of new IP

Abstraction	Time
TLM mode	00:05
Pin Level	00:50
RTL Level	09:15

TLM2.0 (and TLM1.0) designs in hardware though?

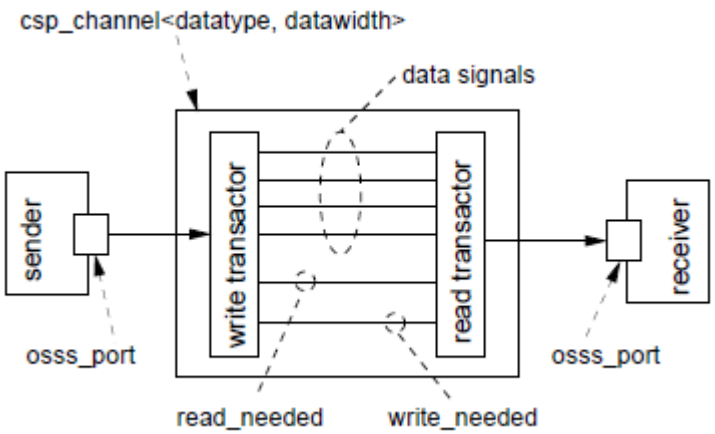


So how to model non-FSA MoC?

- [1] Brunzema & Nebel - Offers a practical way as described

```
void Buffer::send1_wrapper()
{
 byte3 arg;
 wait();       // for reset
 while(true)
 {
  arg = send1_wrapper_arg_in_port->read();
  send1(arg); // this is the original method
  send1_wrapper_sync_out_port->sync_ack();
 }
}
```

```
SC_CTOR(FORK) {
   queryFork = 1;
   SC_CSP_THREAD(addressFork, DP, csp);
};
```



PAR/SEQ – How to synthesize?

- [1] And a body of OCCAM as SystemC…

occam	OSSS
PROC	SC_CTHREAD
CHAN of *datatype*	csp_channel<*datatype*, *datawidth*>
STOP	while(true) wait();
SKIP	; (empty statement)
SEQ	SEQ-transformation
PAR	PAR-transformation
ALT	ALT and ALT_CLAUSE macros

```
PROC buffer(CHAN OF BYTE in,
            CHAN OF BYTE cmd,
            CHAN OF BYTE3 buf.to.filter1,
            CHAN OF BYTE3 buf.to.filter2,
            CHAN OF BYTE3 buf.to.filter3)
  [3]BYTE buffer :    -- internal memory
  BYTE pixel :
  BYTE cmd.code :
  SEQ
    buffer := [ 0, 0, 0 ]
    WHILE TRUE
      ALT
        cmd ? cmd.code
          IF
            cmd.code = clear.cmd
              buffer := [ 0, 0, 0 ]
            TRUE
              SKIP

        in ? pixel
          SEQ
            -- shift in by parallel
            -- assignment
            buffer := [ buffer[1],
                        buffer[2],
                        pixel ]
            PAR
              buf.to.filter1 ! buffer
              buf.to.filter2 ! buffer
              buf.to.filter3 ! buffer
:
```

```
void Buffer::buffer_proc()
{
 unsigned char pixel;
 unsigned char cmd_code;
 byte3 buf;

 buf[0] = buf[1] = buf[2] = 0;

 wait(); // for reset
 while(true)
 {
  ALT
  {
    ALT_CLAUSE(cmd, cmd_code)
    {
     if(cmd_code == CLEAR)
     {
      buf[0] = buf[1] = buf[2] = 0;
     }
    }
    ALT_CLAUSE(in, pixel)
    {
     // shift in with overloaded operator
     buf << pixel;

     {// original form:
      // PAR {
      //       send1(buf);
      //       send2(buf);
      //       send3(buf);
      //     }

      // transformed form:
      // first send the arguments
      send1_wrapper_arg_out_port->write(buf);
      send2_wrapper_arg_out_port->write(buf);
      send3_wrapper_arg_out_port->write(buf);

      // then request synchronisation
      send1_wrapper_sync_in_port->sync_req();
      send2_wrapper_sync_in_port->sync_req();
      send3_wrapper_sync_in_port->sync_req();
    }
   }
  }
 }
}
```

PAR/SEQ – How to synthesize?

- [2] Patel & Shukla - Basically Outlines Extending the Kernel with special CSP threads
  - Model using 'CSPort' rendezvous mechanism, this is perfectly feasible under current simulators

```
1 SC_MODULE(FORK) {
2   int id;
3   int queryFork;
4   CSPnode csp;
5   CSPport<int> fromRight;
6   CSPport<int> fromLeft;
7   int * drop;
8   int * pick;
9
10  ProcInfo proc;
11
12  void reqFork();
13  void addressFork();
14
15  SC_CTOR(FORK) {
16    queryFork = 1;
17    SC_CSP_THREAD(addressFork, DP, csp);
18  };
19 };
```

  - CSPnode however relies on specific threading, this could be implemented using a FSM master THREAD/CTHREAD
  - So a model needs to be written to support anything other than FSA behavior without major architecture

[3] Describes the situation quite well

In the recent years we have seen the evolution of system level design frameworks and languages such as SystemC and SpecC [8, 15]. Recent efforts in raising the level of abstraction in hardware description languages beyond the RTL is exemplified by SystemVerilog [16] and Heterogeneous Extensions for SystemC [9]. However, the success of any of these languages is predicated upon the adoption of the language as a standard design entry language by the industry resulting in a closure of the productivity gap. SystemC is an example of such a language. In fact, adding new libraries to SystemC such as mixed-model analog-digital signal libraries (expected in SystemC 4.0) and other libraries alone, are insufficient in making SystemC as a standard design entry language. Instead, SystemC must introduce heterogeneity and behavioral hierarchy in addition to the inclusion of special purpose libraries.

Summary

- Chip Design has become increasingly complex, and people are now looking to new ways for designing
- Whilst pure MOC may seem esoteric to the layman, if properly introduced they can provide elegant ways to describe more complex models
- Non-Mainstream languages, HandelC/SpecC/..., face an up-hill battle as conventional wisdom questions their uses.
- Current synthesizable languages focus heavily on supporting tradition whilst balancing the need to move to a higher level of abstraction. (SystemC, SystemVerilog)
- Process Control Extensions suggesting fork/join/rejoin are being proposed so that in the near future a more fully synthesizable approximation to a more full heterogeneous MoC environment should be possible with less agitation

References

- [1] Brunzema & Nebel – CSP with Synthesizable SystemC and OSSS
- [2] Patel & Shukla – SystemC Kernel Extensions for Heterogeneous System Modeling
- [3] hiren, mathaikutty, shukla – Implementing Multi-MoC Extensions for SystemC: Adding CSP & FSM Kernels for Heterogeneous Modeling
- [4] Mentor Graphics – HLS Blue Book