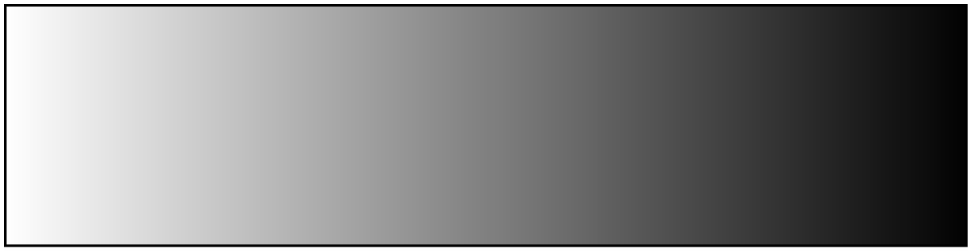
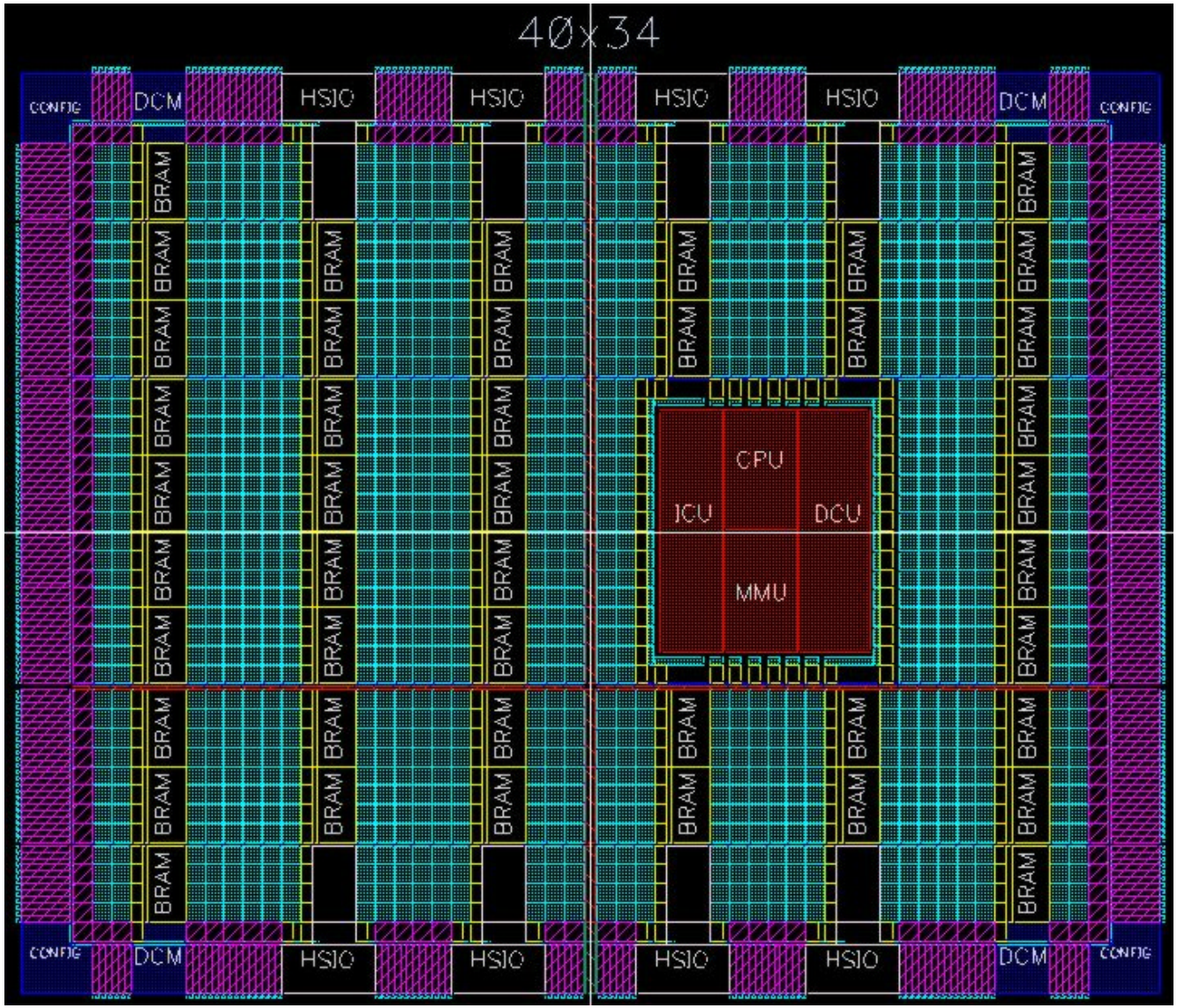
Co-Design and Co-Verification using a Synchronous Language

Satnam Singh

Xilinx Research Labs

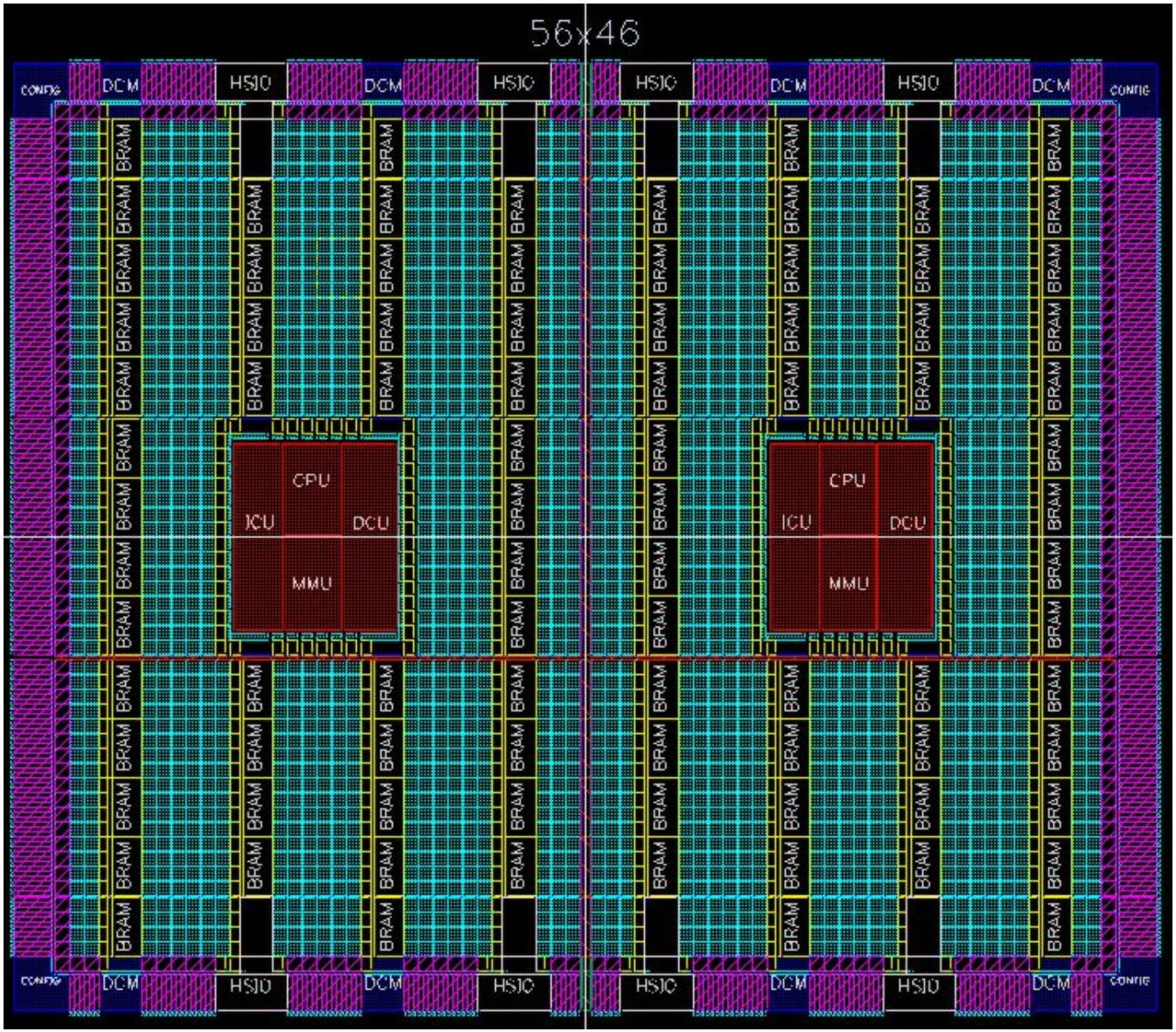


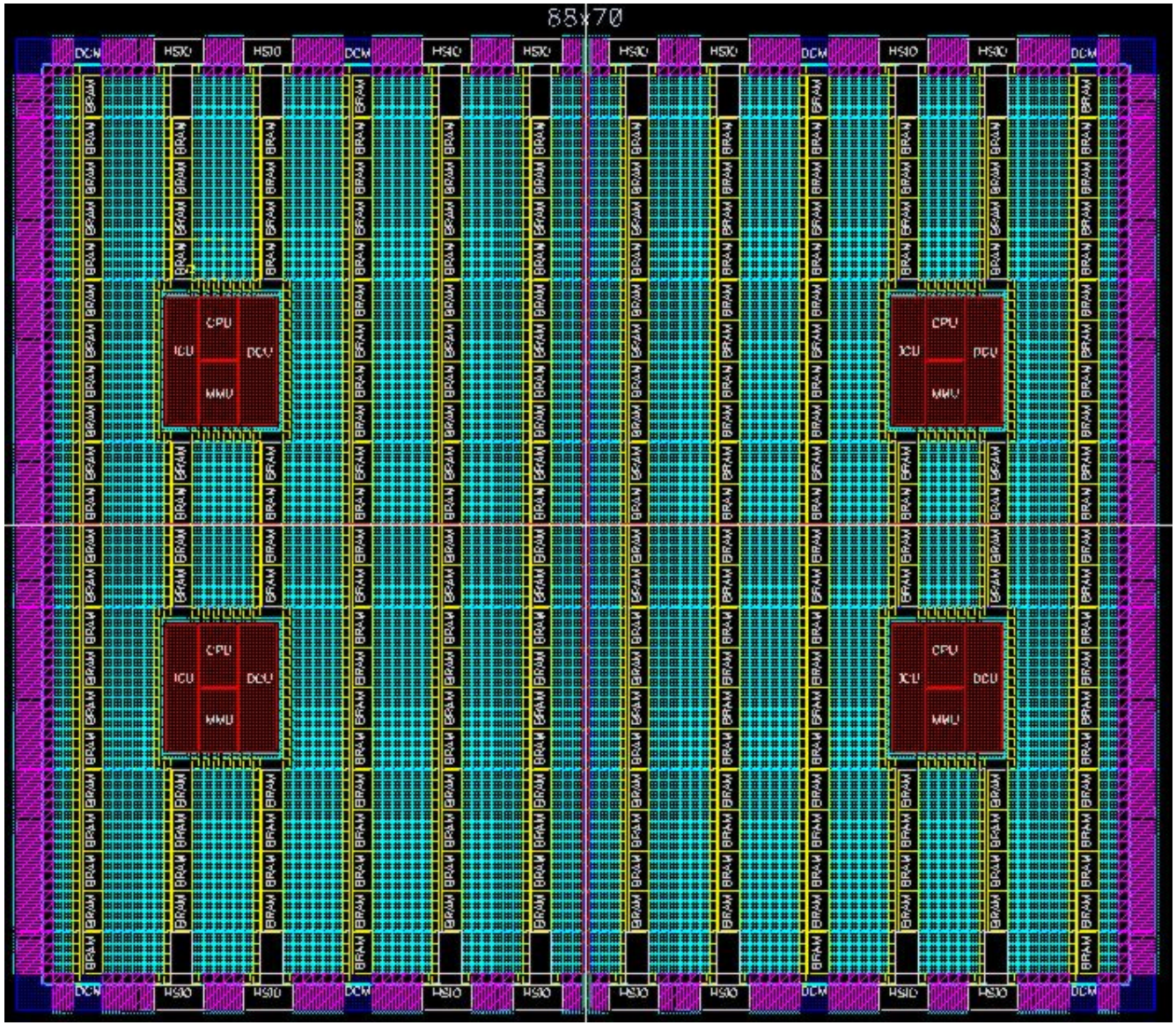
Software Hardware

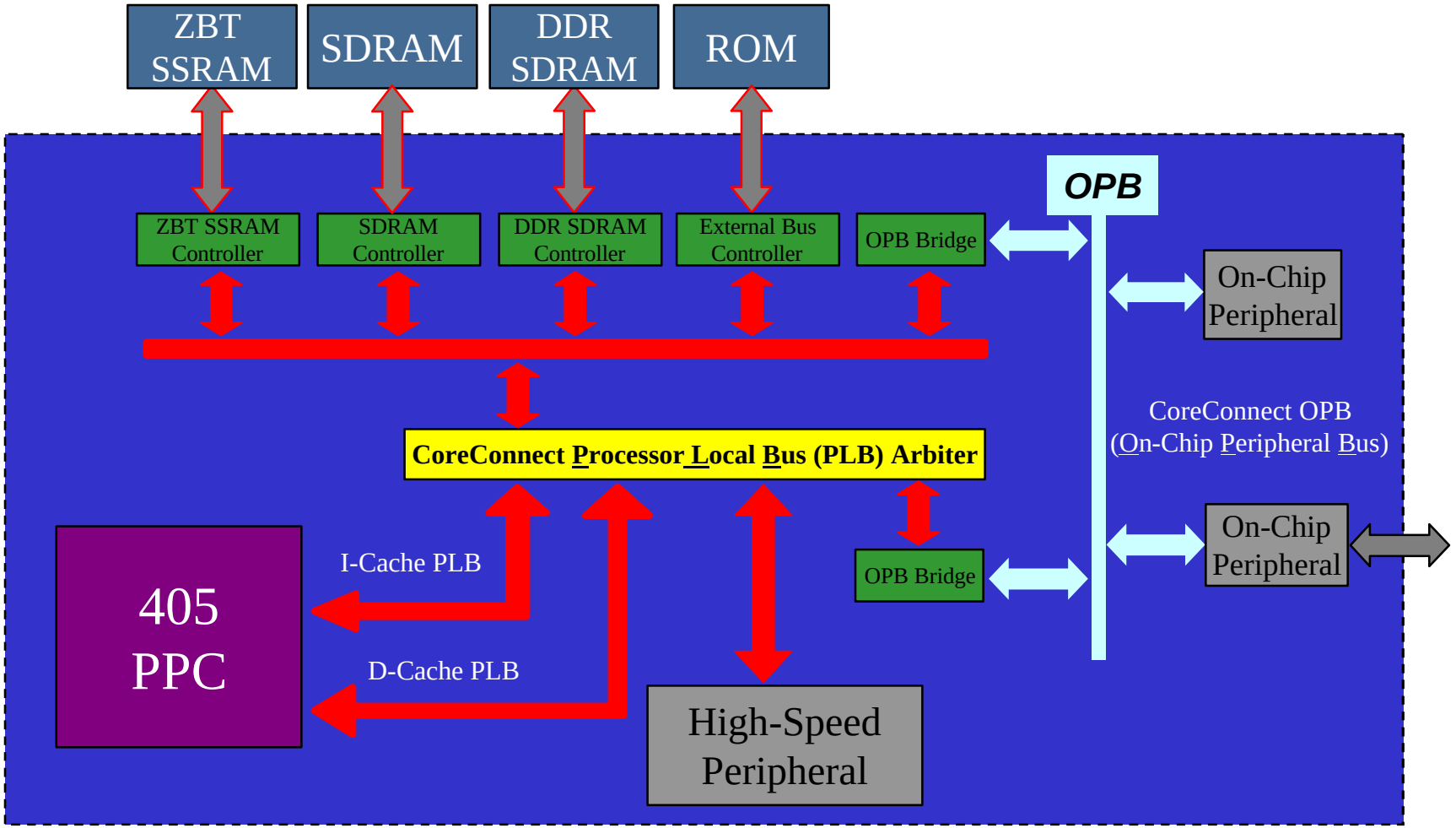

40x34
CONFIG
DCM
HSIO
HSIO
HSIO
HSIO
DCM
CONFIG
BRAM
CPU
ICU
DCU
MMU
CONFIG
DCM
HSIO
HSIO
HSIO
HSIO
DCM
CONFIG

Virtex-II PRO

Device	Array Size	Logic Gates	PPCs	GBIOs	BRAMs
2VP2	16 x 22	38K	0	4	12
2VP4	40 x 22	81K	1	4	28
2VP7	40 x 34	133K	1	8	44
2VP20	56 x 46	251K	2	8	88
2VP50	88 x 70	638K	4	16	216


56x46
CONFIG
DCM
HSIO
CPU
ICU
DCU
MMU
BRAM


88x70
DCM
HSIO
BRAM
CPU
ICU
DCU
MMU


ZBT SSRAM
SDRAM
DDR SDRAM
ROM
ZBT SSRAM Controller
SDRAM Controller
DDR SDRAM Controller
External Bus Controller
OPB Bridge
OPB
On-Chip Peripheral
CoreConnect OPB (On-Chip Peripheral Bus)
CoreConnect Processor Local Bus (PLB) Arbiter
405 PPC
I-Cache PLB
D-Cache PLB
OPB Bridge
On-Chip Peripheral
High-Speed Peripheral

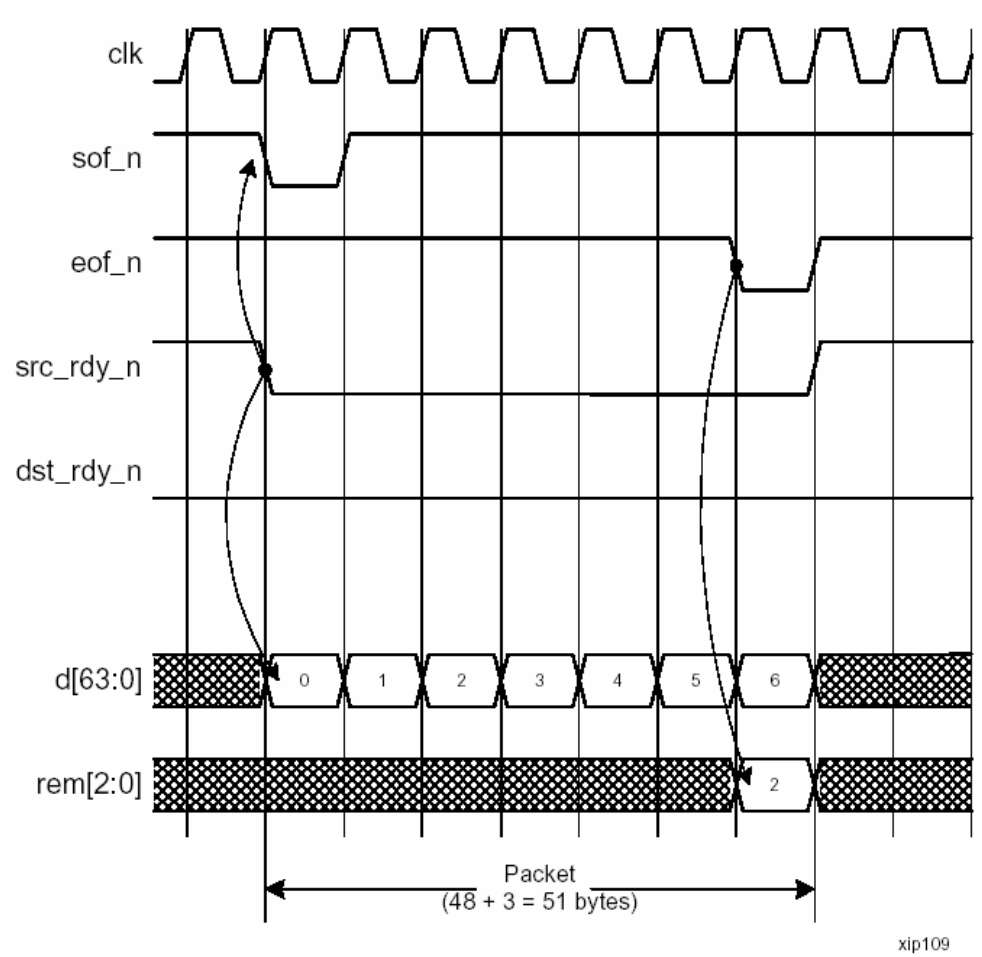
Formal Techniques Project

- Domain specific languages for hardware design and verification (Lava) and cryptology (Cryptol).
- Formal methods for CAD (routing) and dynamic reconfiguration (reconfiguration controllers).
- **Formal notations/representations for HW/SW co-design and verification (e.g. Esterel)**
- Property checking (PSL/Sugar)
- IP-reuse (more powerful type systems la de Alfaro and Henzinger)

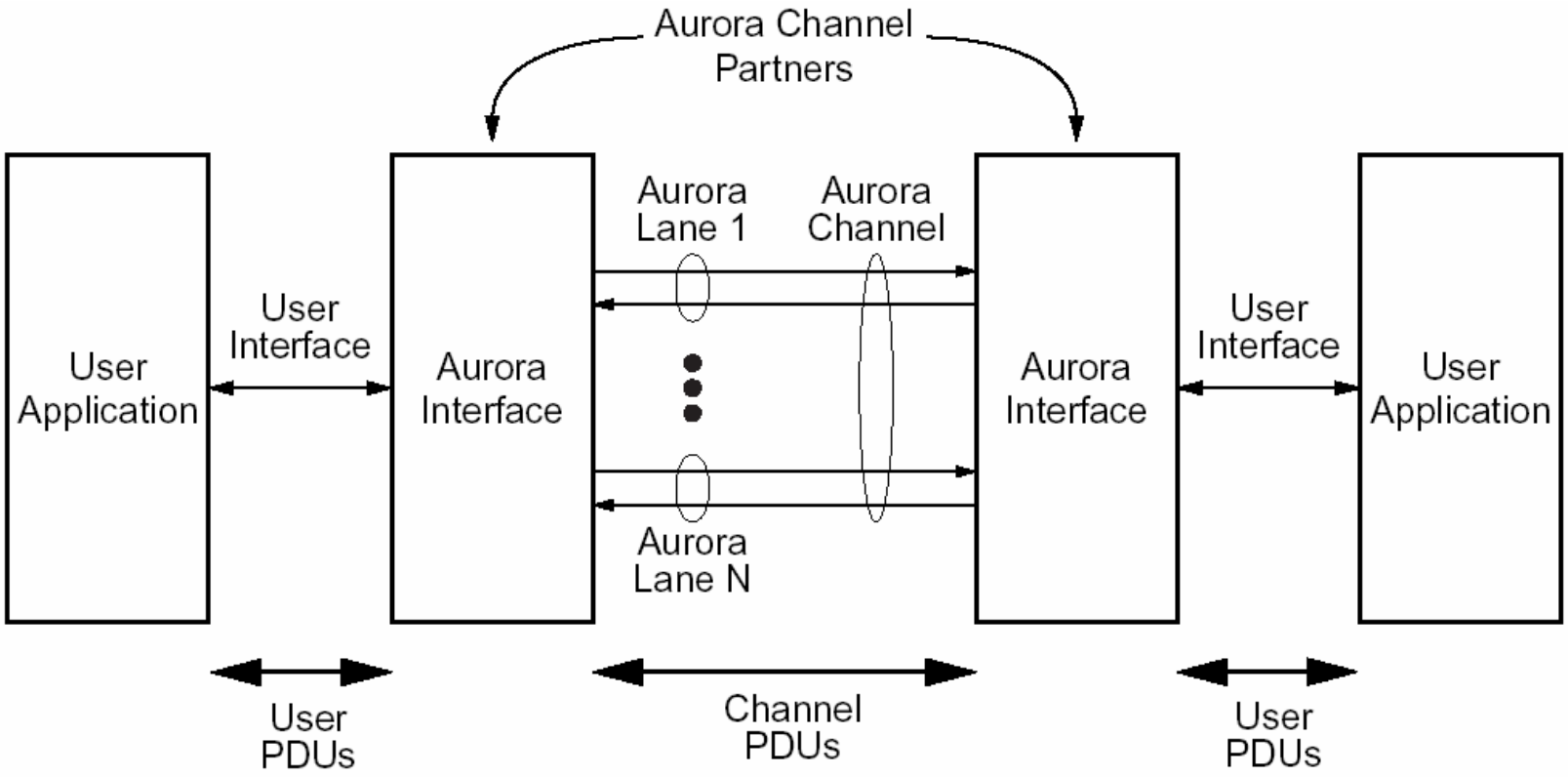
~~Problems~~ Challenges

- Customer requirements for migrating software into hardware ("salmon effect"):
  - determinism: multiple SW processes on RTOS vs. genuinely concurrent HW
  - verification
  - isolation
- Customer requirements to trade-off HW/SW partitioning for products at different price points.
- Requirements for verification of SW+HW
- Safe Dynamic Reconfiguration
- Verification of control-based systems

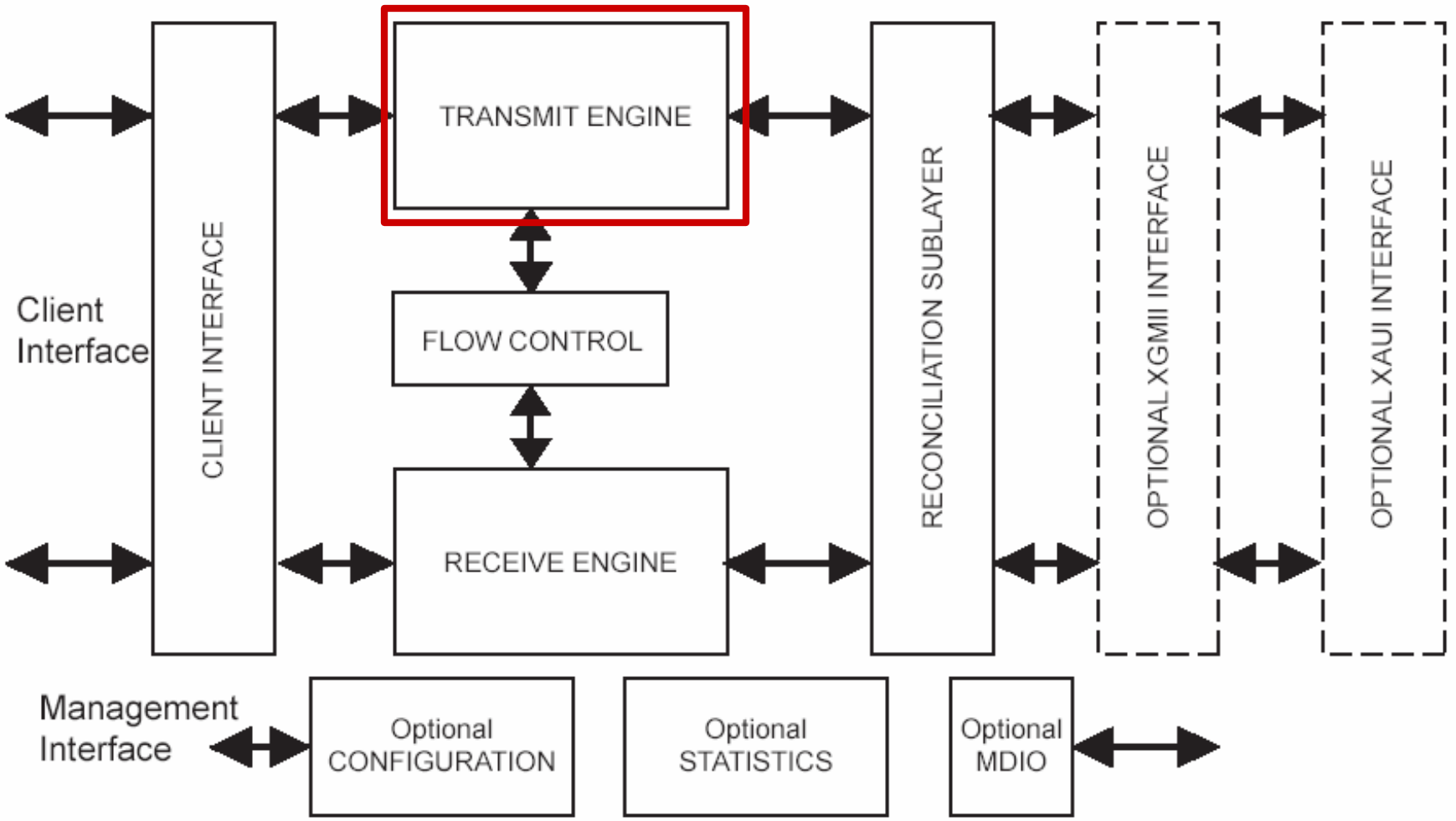
LocalLink (Point to Point)



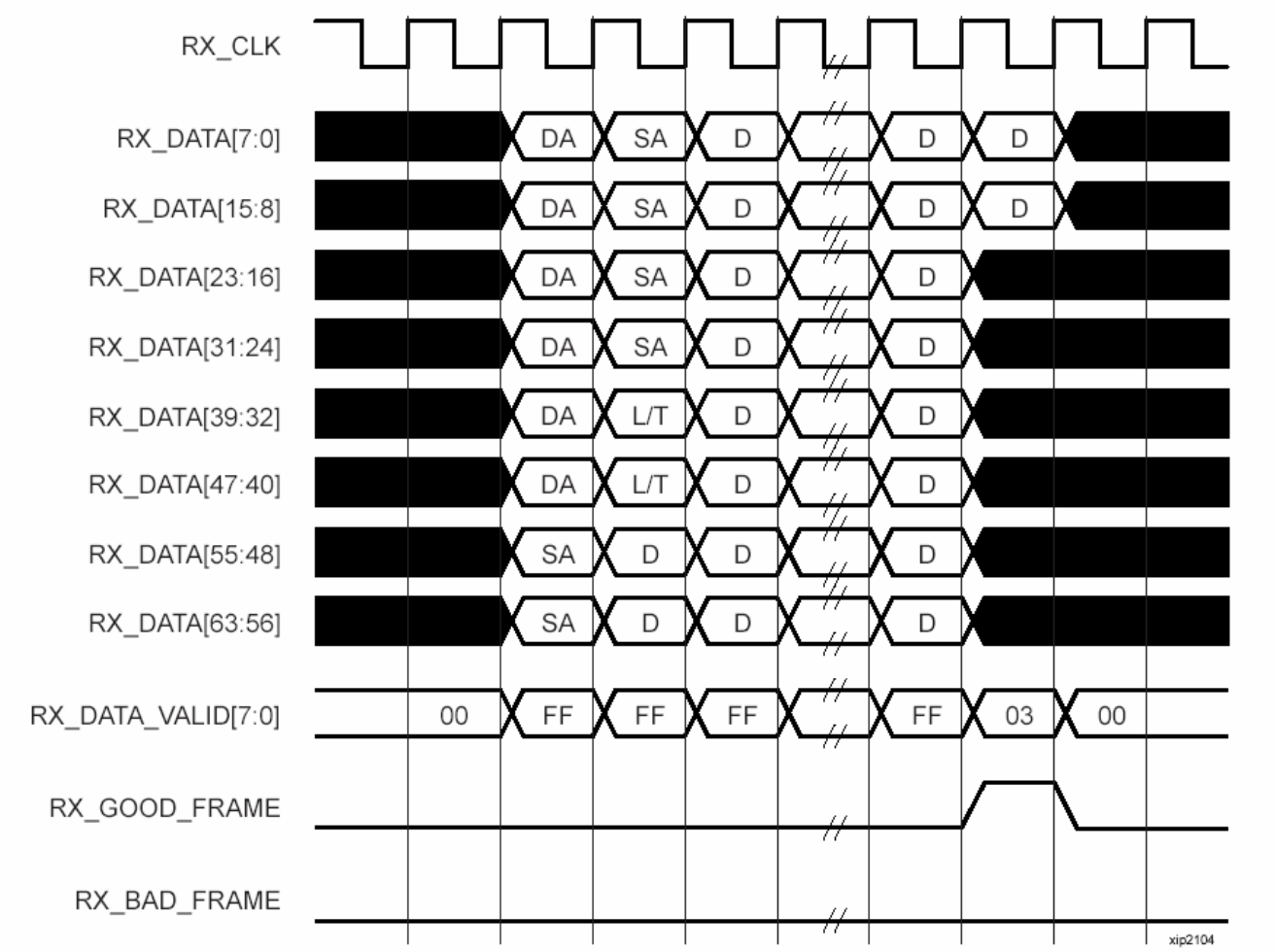
Aurora (Link Layer Protocol)



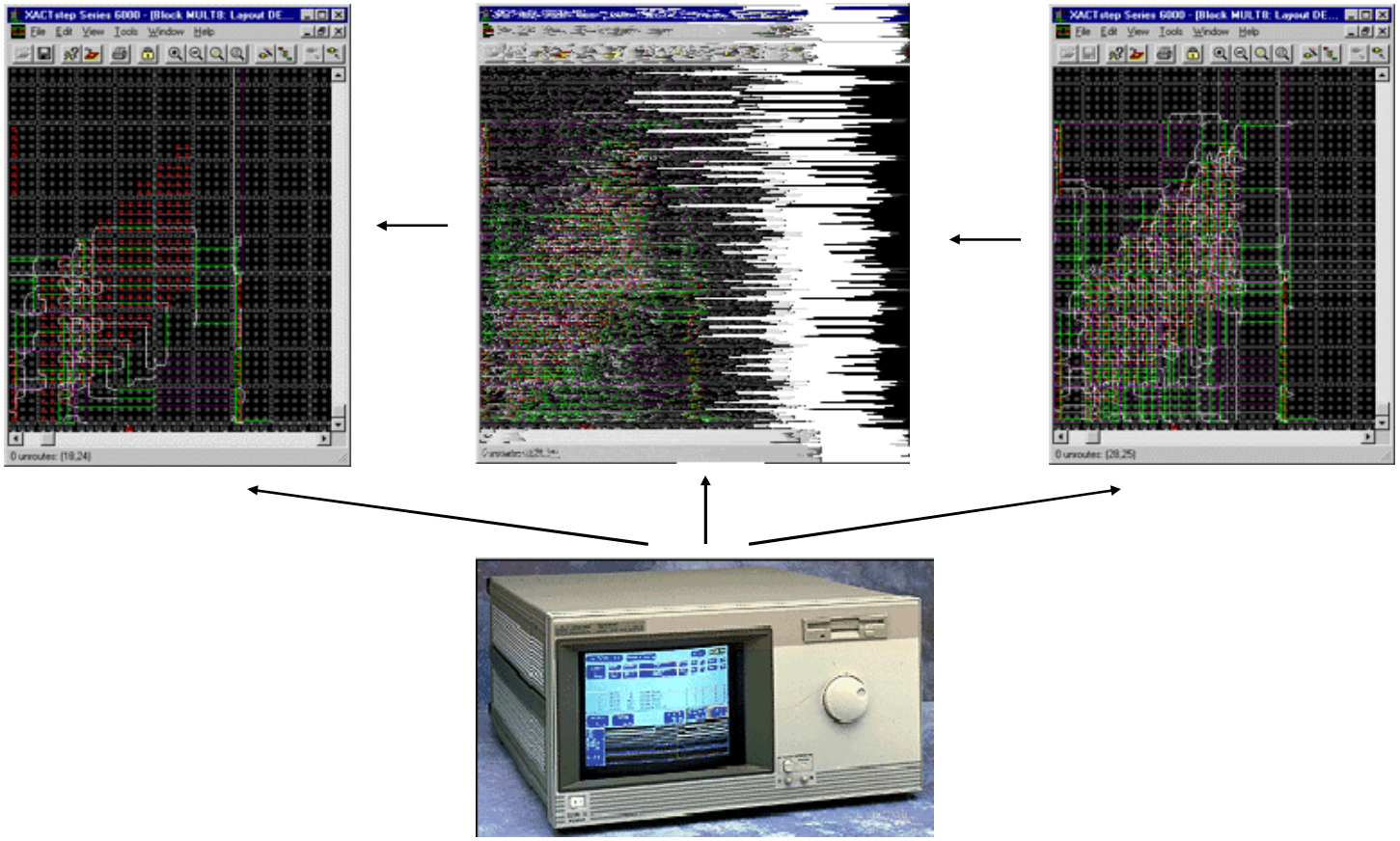
TX of 10 Gigabit Ethernet MAC



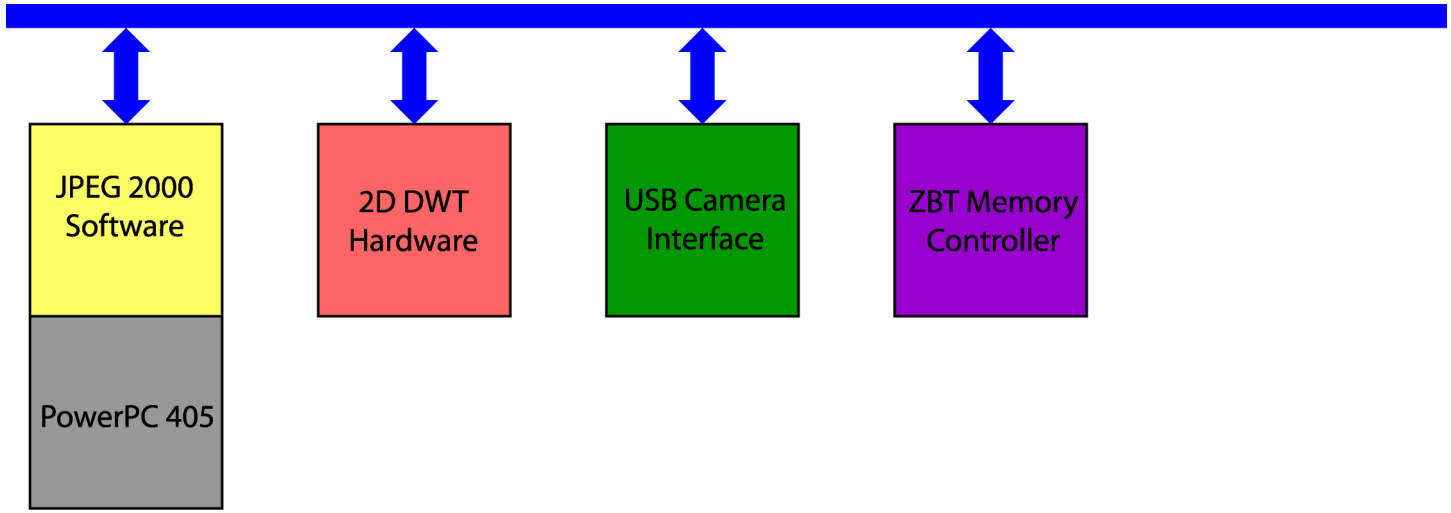
Verify RX Control Signals



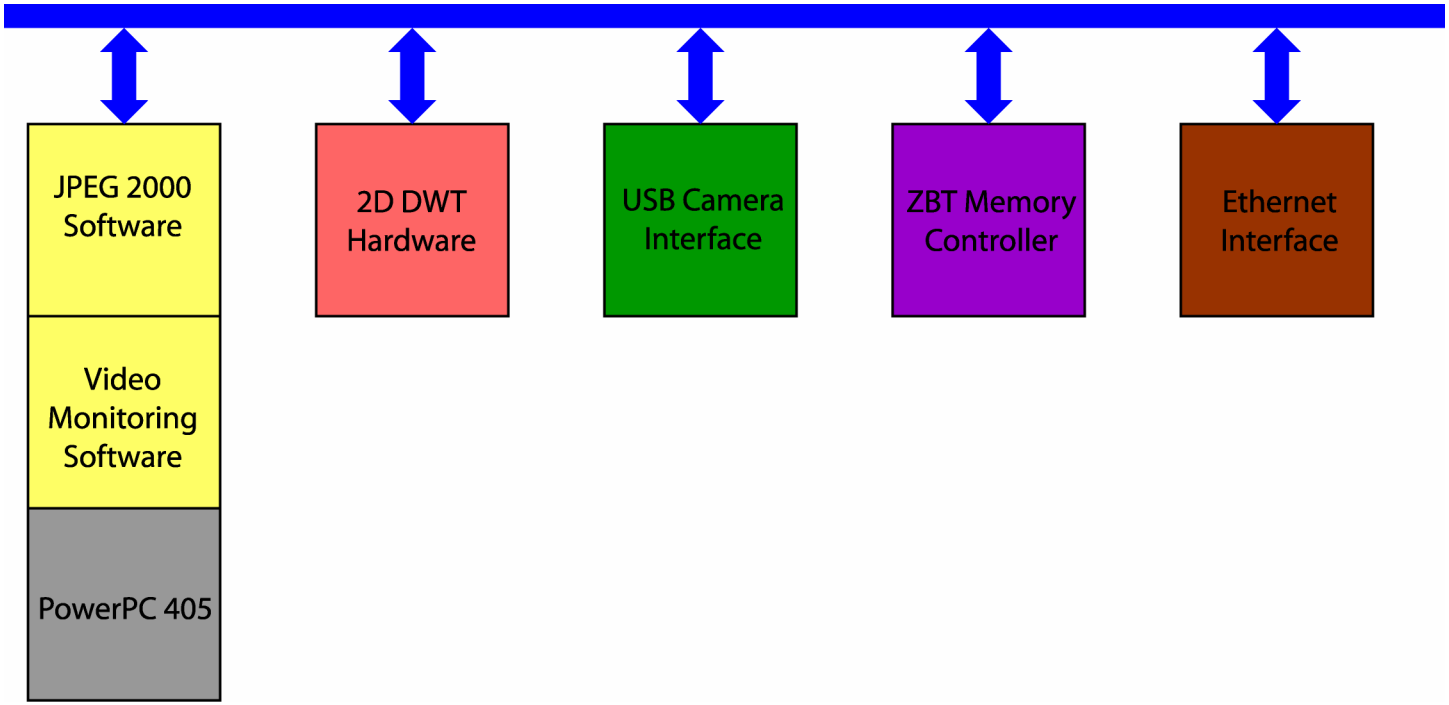
Safe Dynamic Reconfiguration



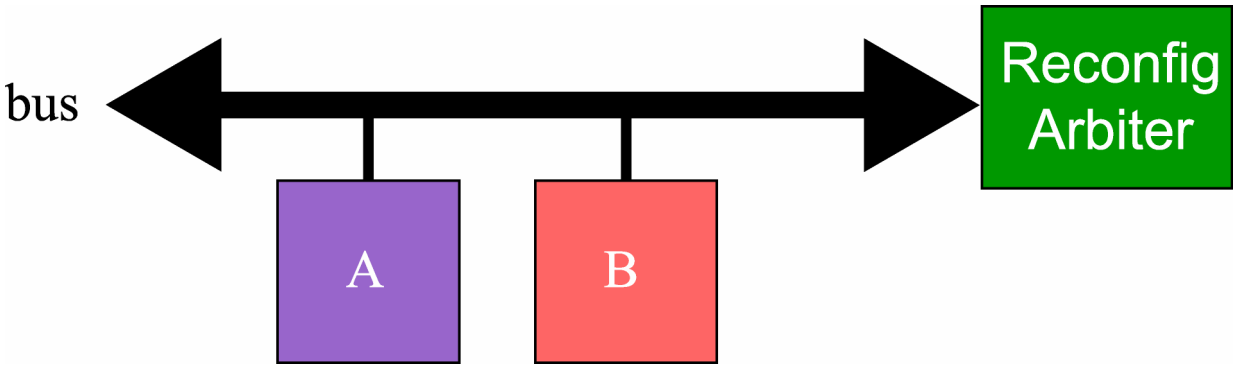
JPEG2000 Platform



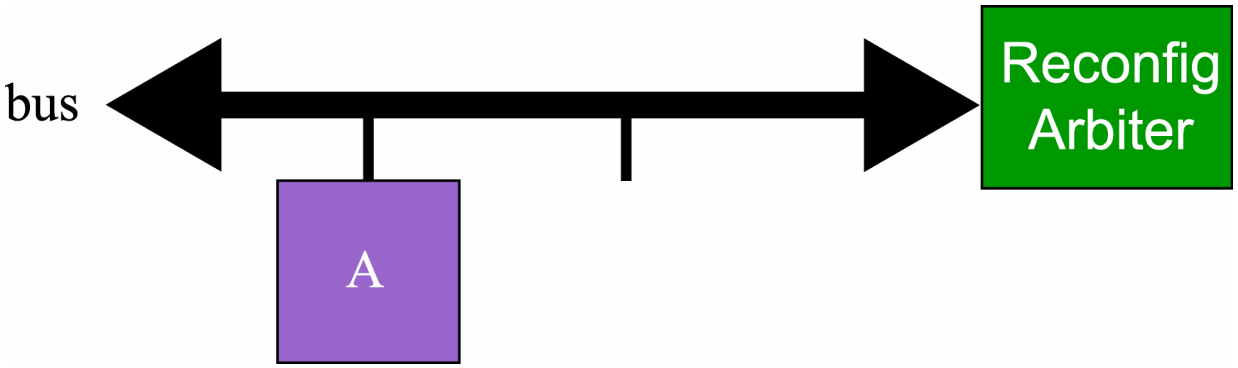
Video Monitoring Application



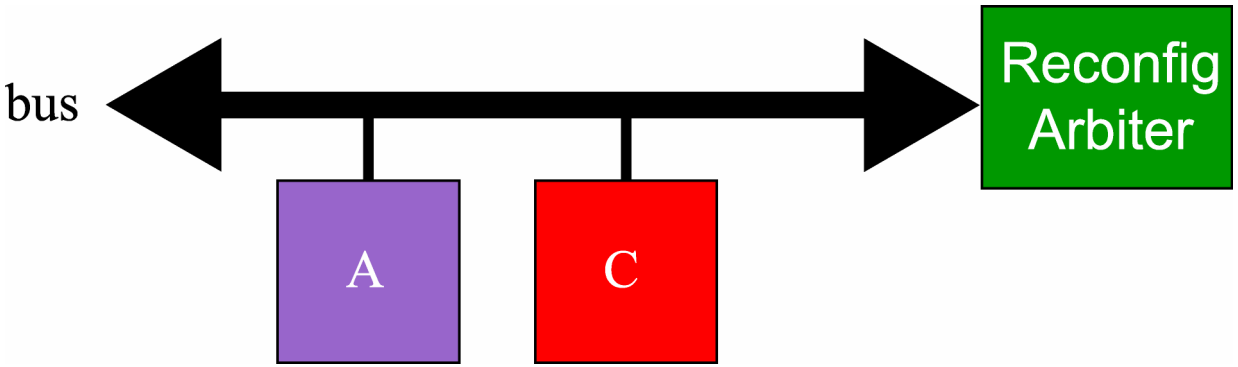
Bus Based Reconfiguration



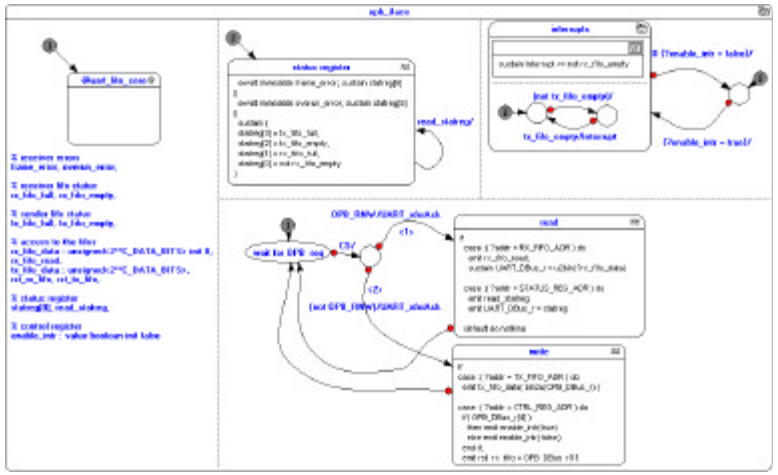
Bus Based Reconfiguration



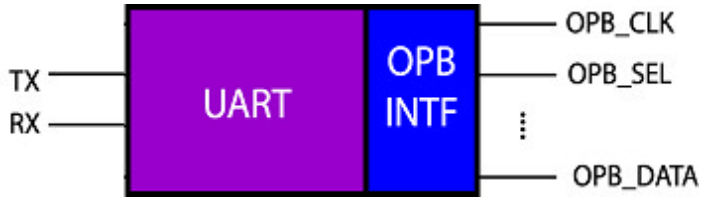
Bus Based Reconfiguration



Single Specification for Hardware and Software



HW/SW agnostic specification



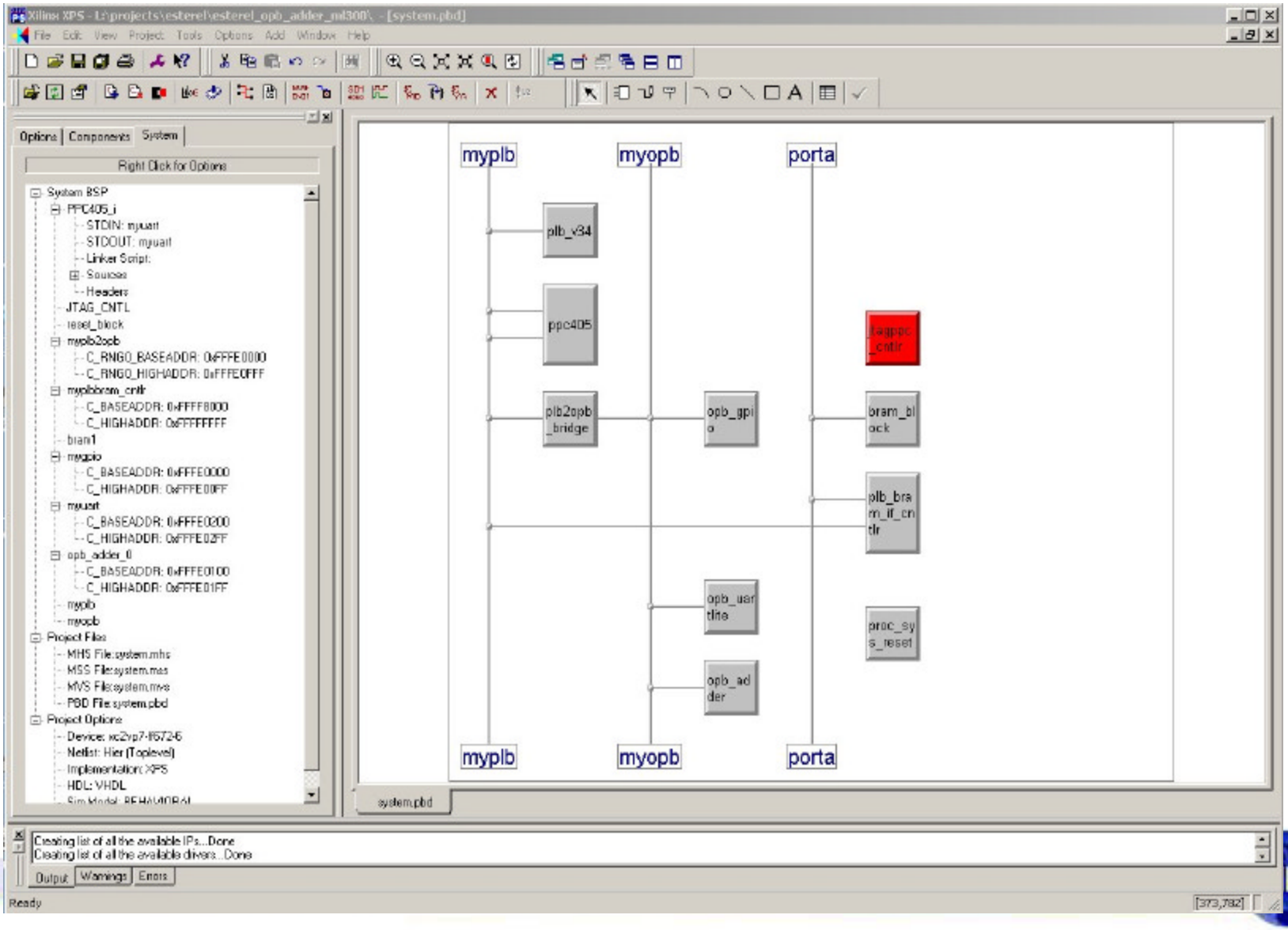
VHDL, Verilog -> hardware implementation

```
void uart_device_driver ()
{
.....
}
```

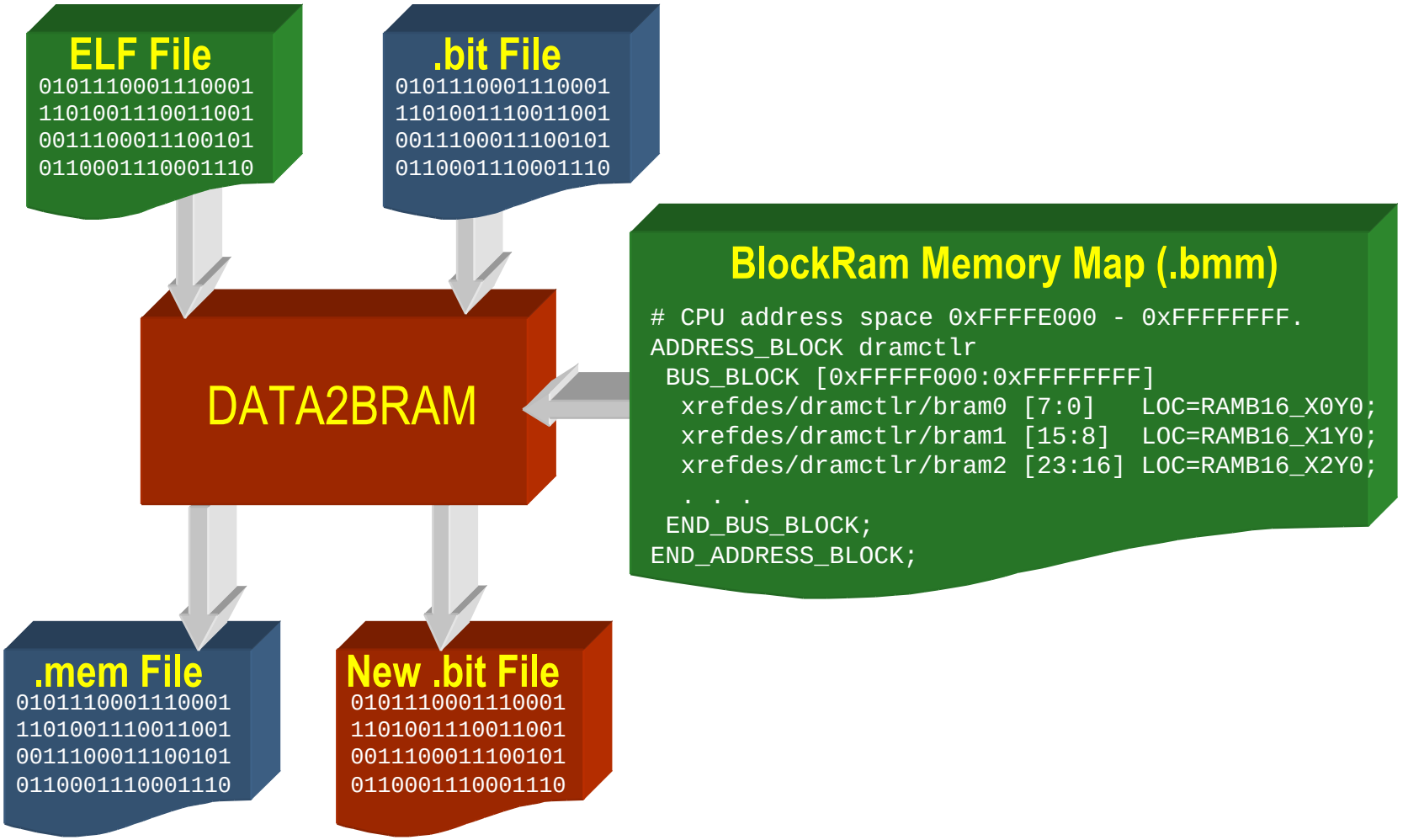
uart.c

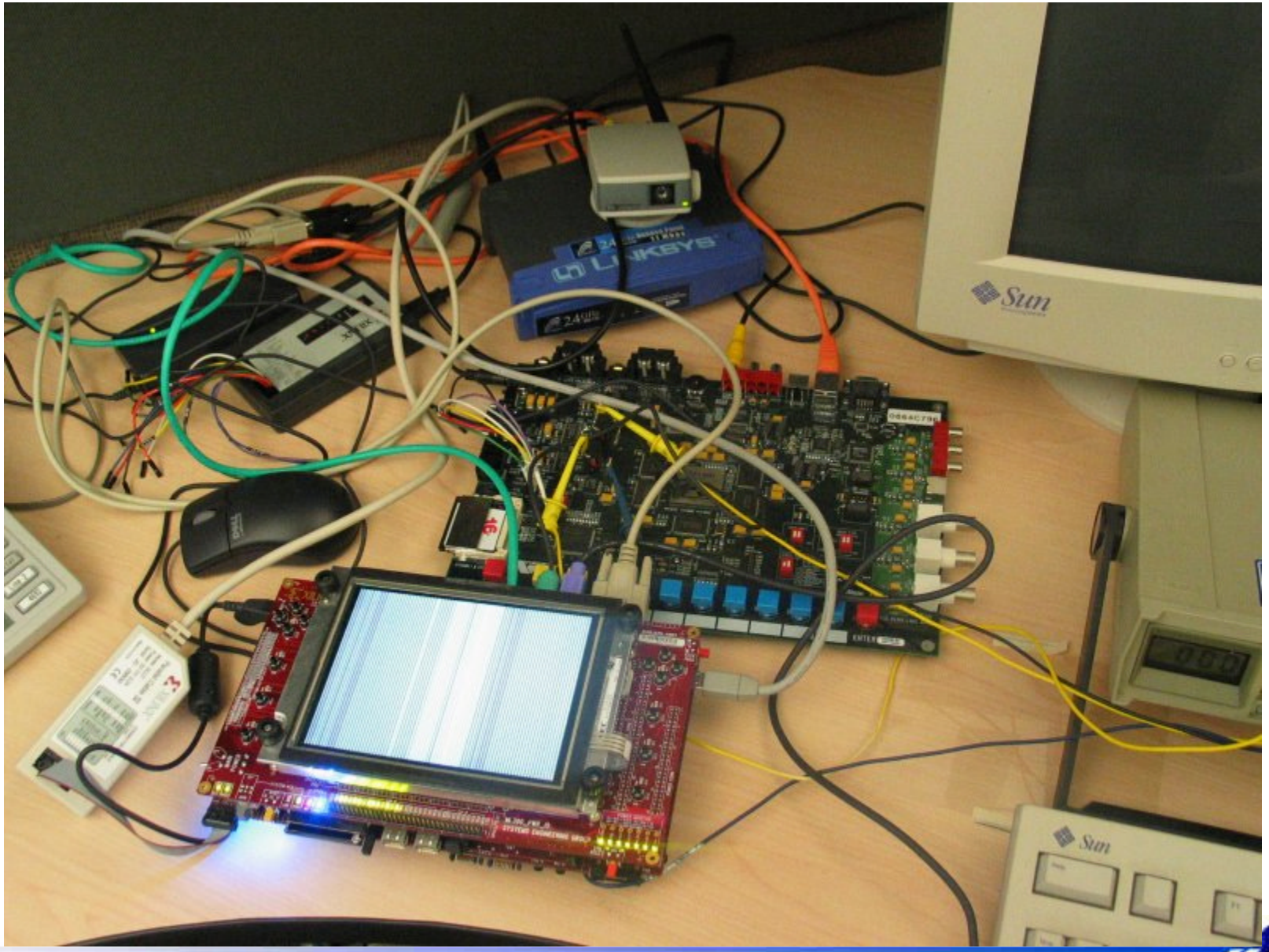
C -> software implementation

Embedded Developer Kit



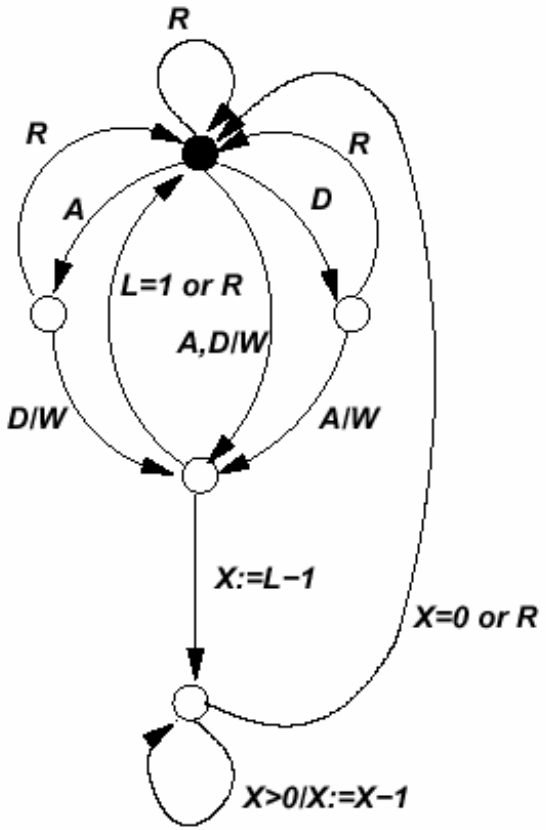
Configuration




LINKSYS
Sun
Sun

FSM Specification

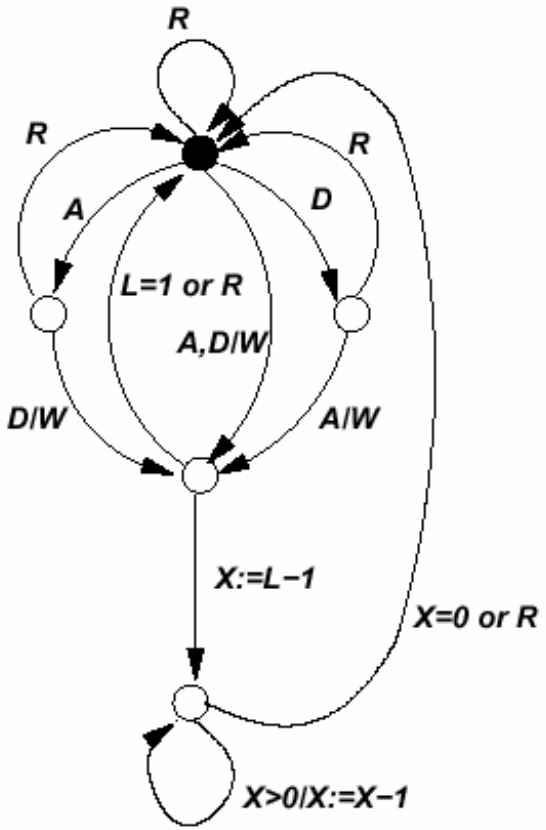
Write to memory as soon as `Addr` *and* `Data` *have arrived. Wait for memory* `Latency` *before iterating. Restart behavior each* `Replay`.



Esterel Specification

Write to memory as soon as `Addr` *and* `Data` *have arrived. Wait for memory* `Latency` *before iterating. Restart behavior each* `Replay`.

```
loop
   abort
      { await Addr || await Data };
      emit Write(funcW(?Addr,?Data));
      await Latency tick
   when Replay
end loop
```



FIFO Extract

```
{ // equation style
  sustain{
    ReadEmpty = Read and pre(FifoIsEmpty)
    FifoIsEmpty = if (?Entries = 0) }
||
  // imperative style
  every DeltaEntries do
    emit next ?Entries <= ?Entries
                        + ?DeltaEntries;
  end every
}
```

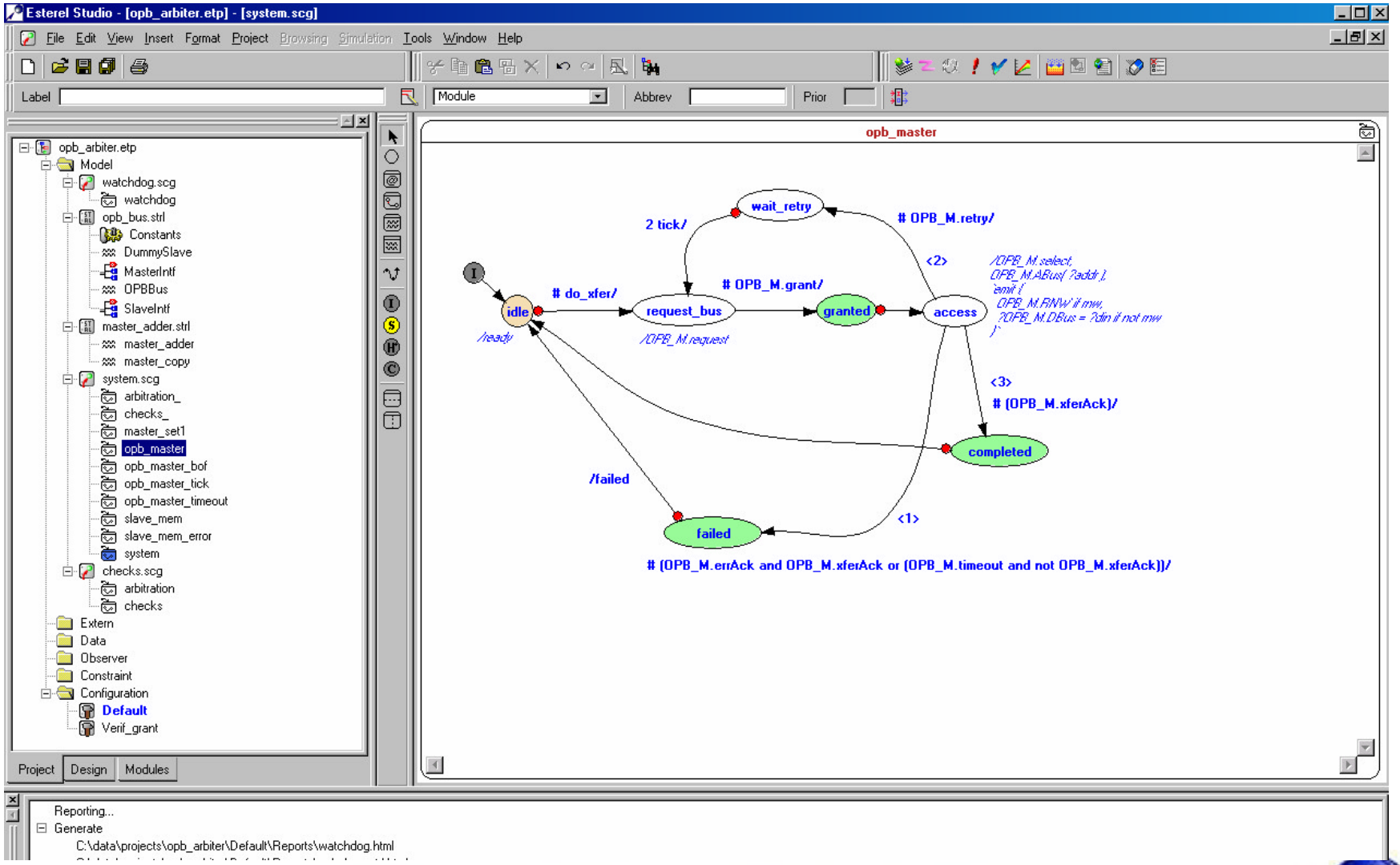
Concurrent Loops

```
for i < 2 dopar
  sustain {
    FifoEmpty[i] = if (pre(?Entries) = i)
    FifoFull[i]  = if (pre(?Entries) =
                       Size - i)
  }
end for
```

Orthogonality

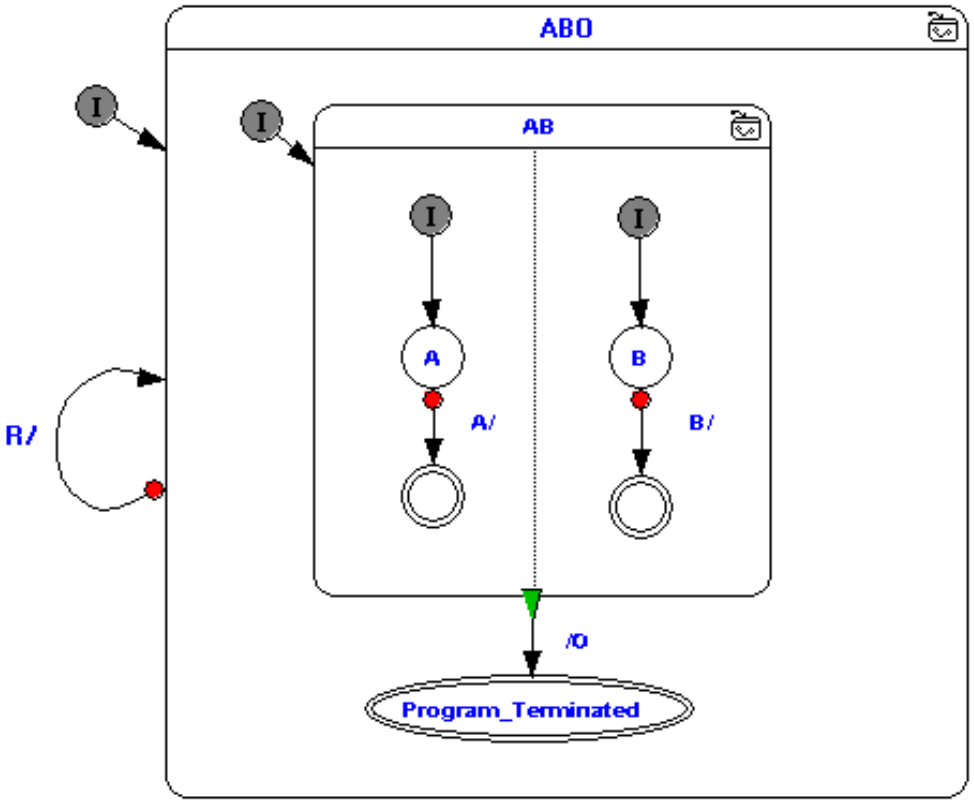
- Orthogonal language constructs for:
  - Sequencing
  - Concurrency
  - Waiting
  - Pre-emption
- Freely mixable at any level.
- “Things are only written once.” Gérard Berry.

Esterel Studio



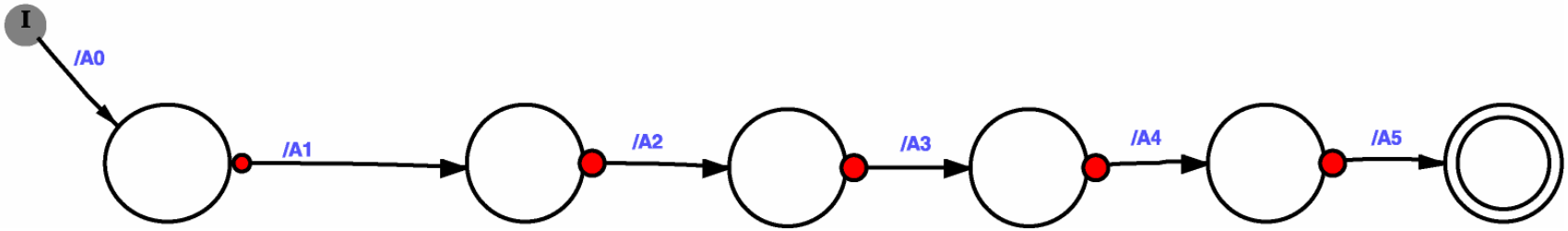
Creating design

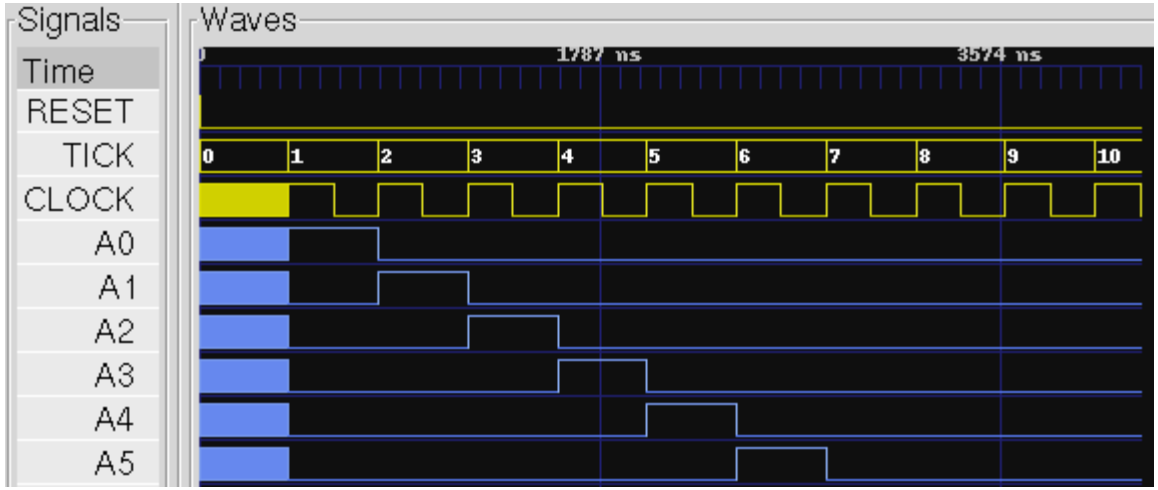
Via Safe State Machines

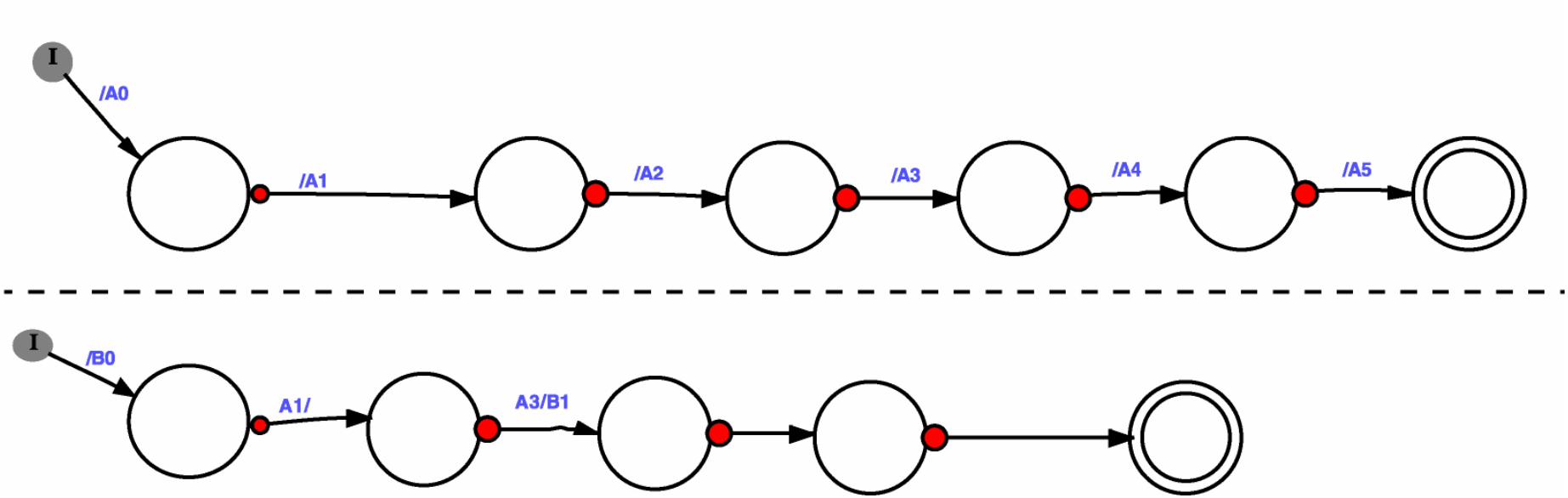


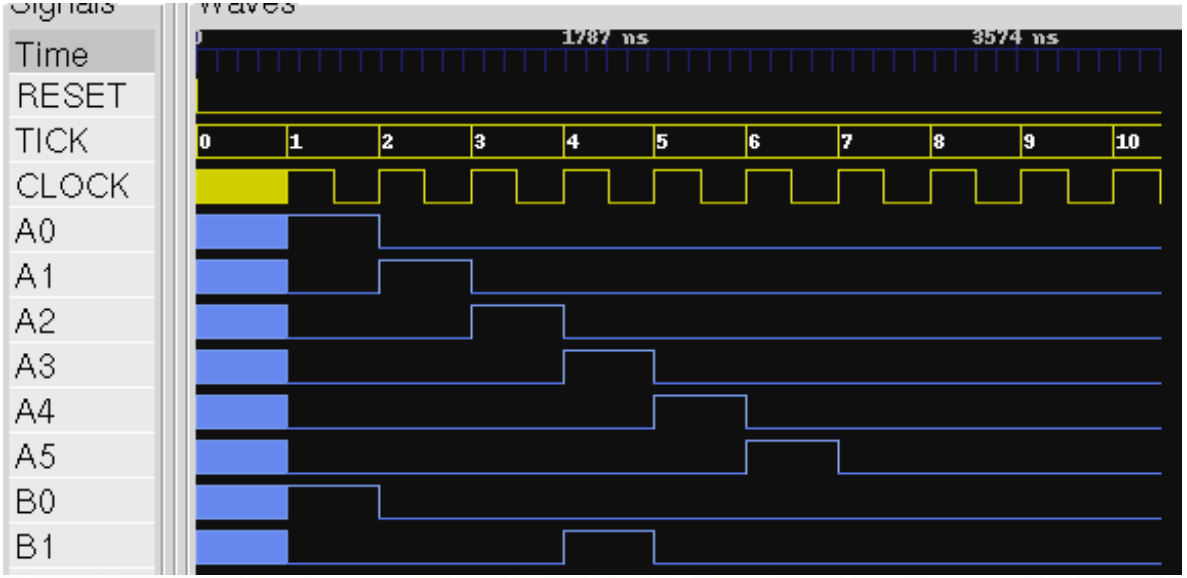
Via Esterel code

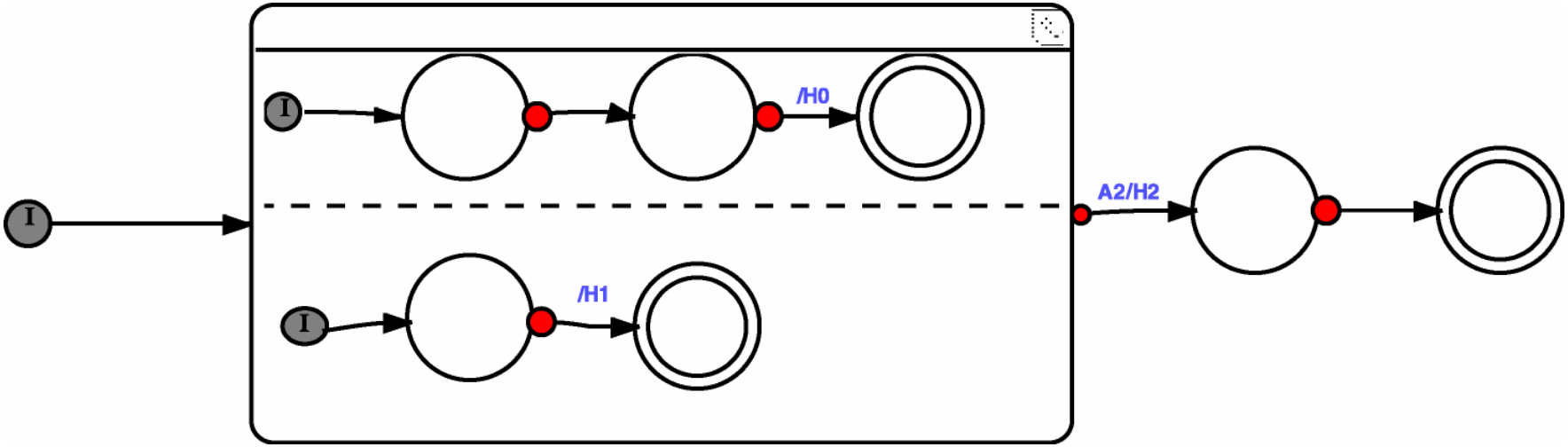
```
loop
   [ await A  ||  await B ] ;
   emit O
each  R
```

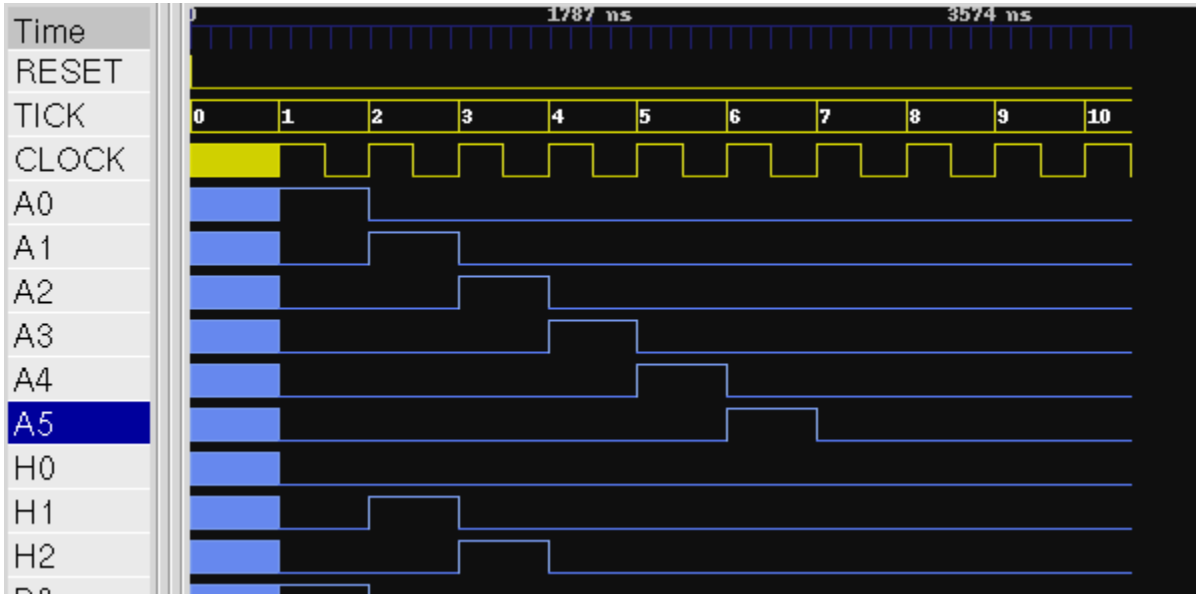

I
/A0
/A1
/A2
/A3
/A4
/A5

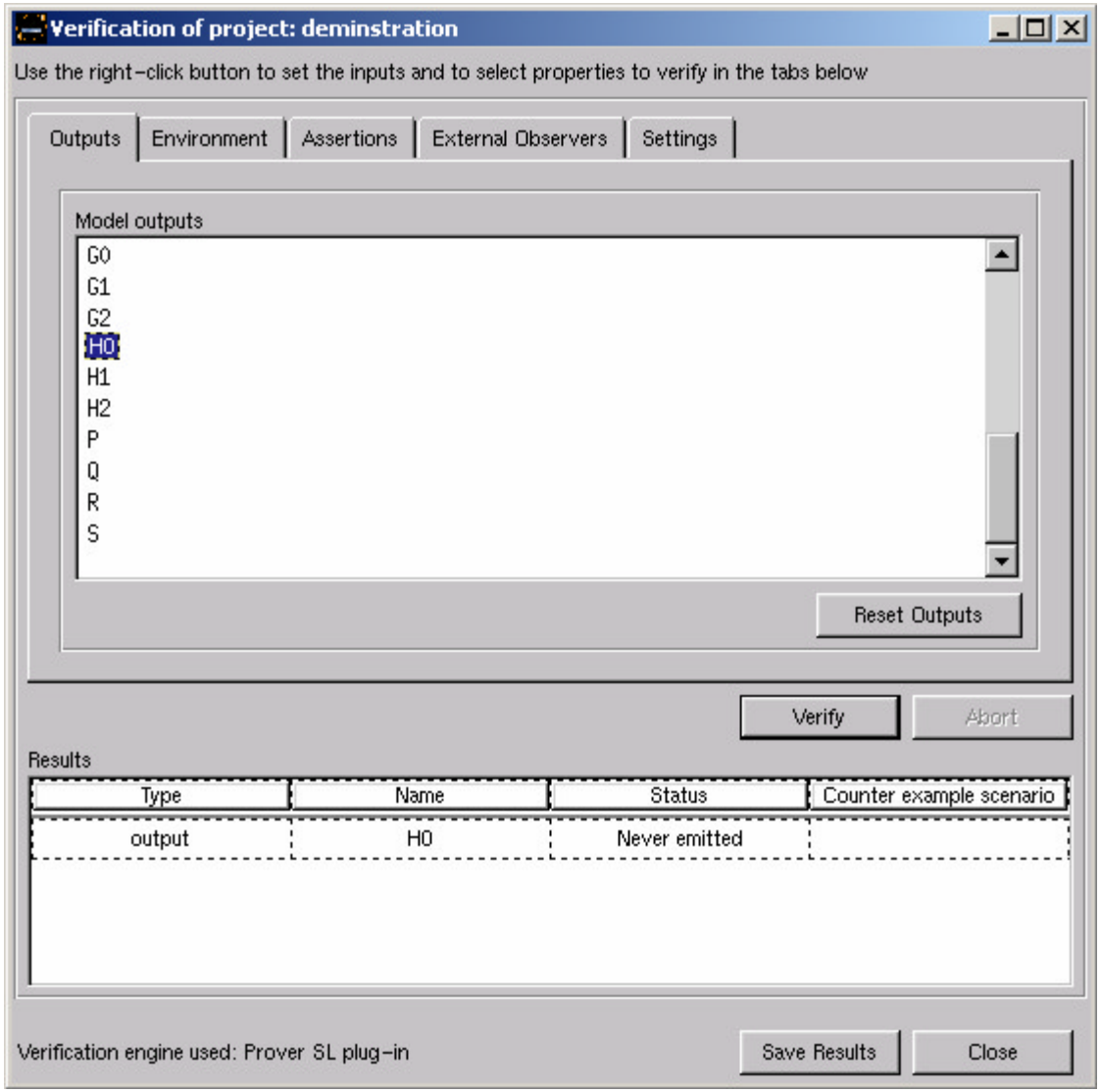

Signals
Waves
Time
RESET
TICK
CLOCK
A0
A1
A2
A3
A4
A5
1787 ns
3574 ns
0
1
2
3
4
5
6
7
8
9
10


I
/A0
/A1
/A2
/A3
/A4
/A5
I
/B0
A1/
A3/B1

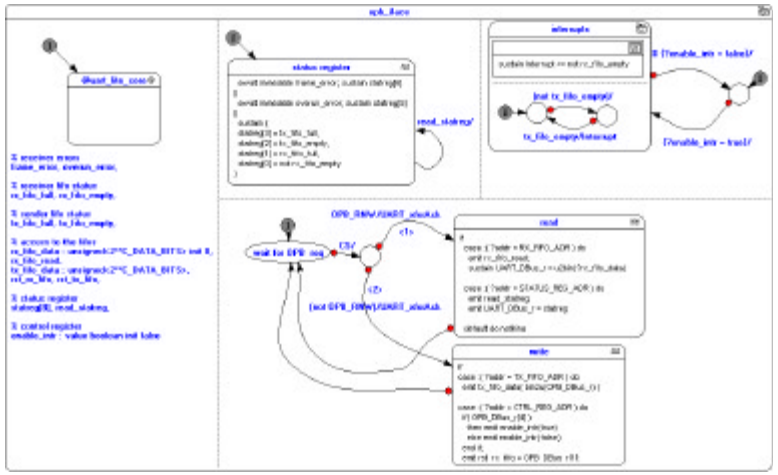

Signals
Waves
Time
RESET
TICK
CLOCK
A0
A1
A2
A3
A4
A5
B0
B1
1787 ns
3574 ns
0
1
2
3
4
5
6
7
8
9
10


I
I
/H0
I
/H1
A2/H2

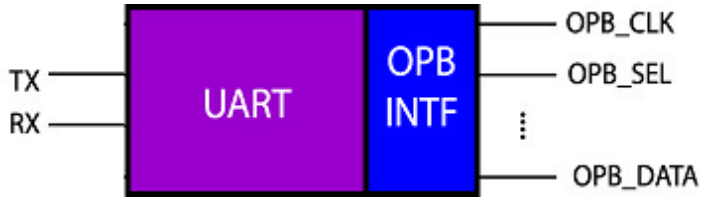

Time
RESET
TICK
CLOCK
A0
A1
A2
A3
A4
A5
H0
H1
H2
1787 ns
3574 ns
0
1
2
3
4
5
6
7
8
9
10


Verification of project: deminstration
Use the right-click button to set the inputs and to select properties to verify in the tabs below
Outputs
Environment
Assertions
External Observers
Settings
Model outputs
G0
G1
G2
H0
H1
H2
P
Q
R
S
Reset Outputs
Verify
Abort
Results
Type
Name
Status
Counter example scenario
output
H0
Never emitted
Verification engine used: Prover SL plug-in
Save Results
Close

Code generation



Esterel design



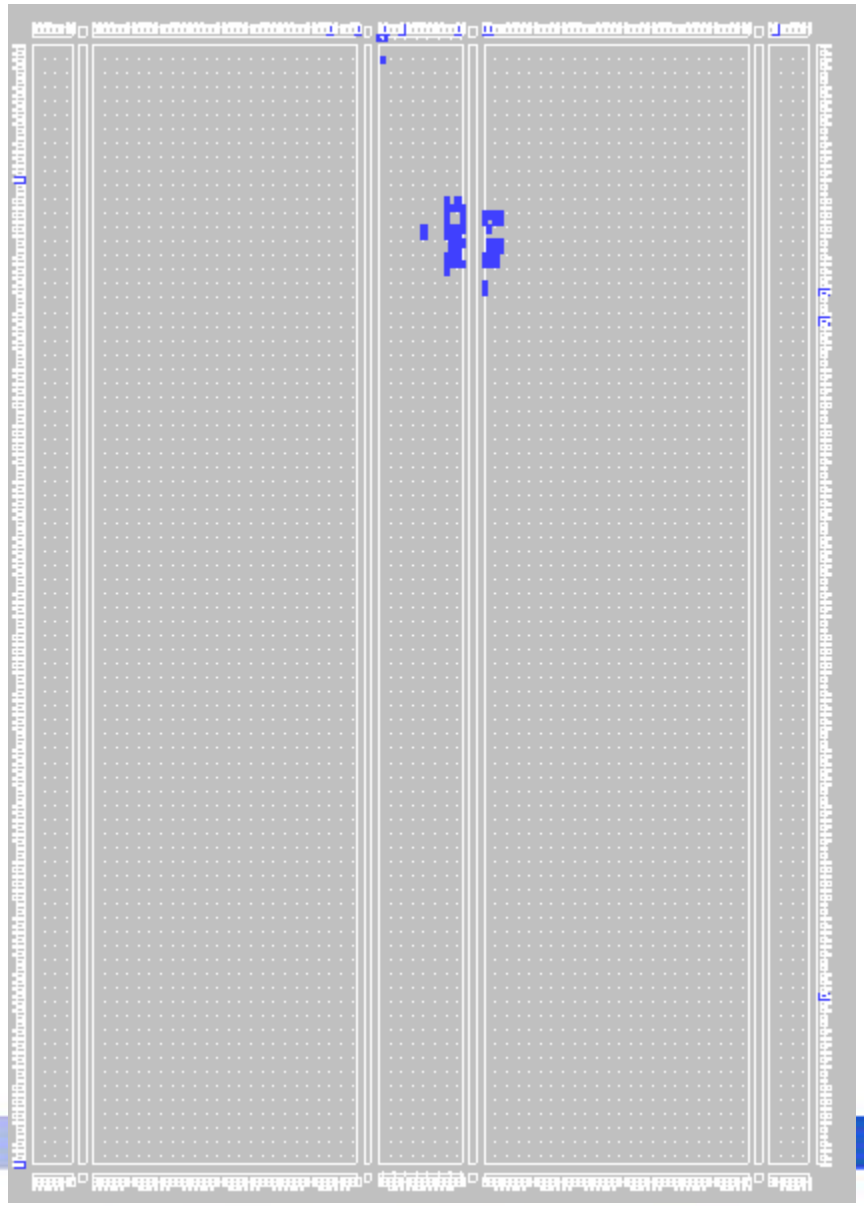
VHDL, Verilog -> hardware implementation

```
void uart_device_driver ()
{
.....
}
```

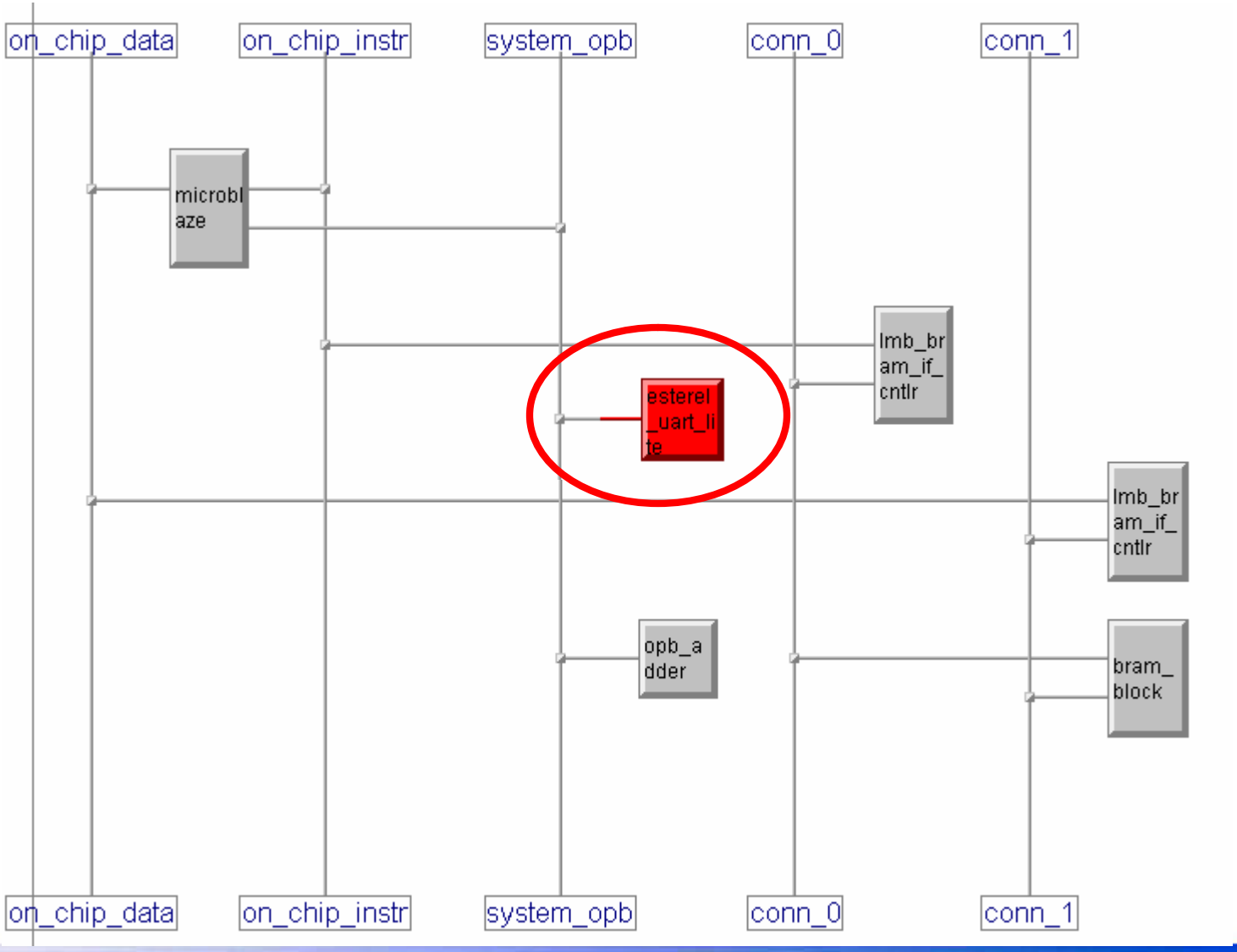
uart.c

C -> software implementation

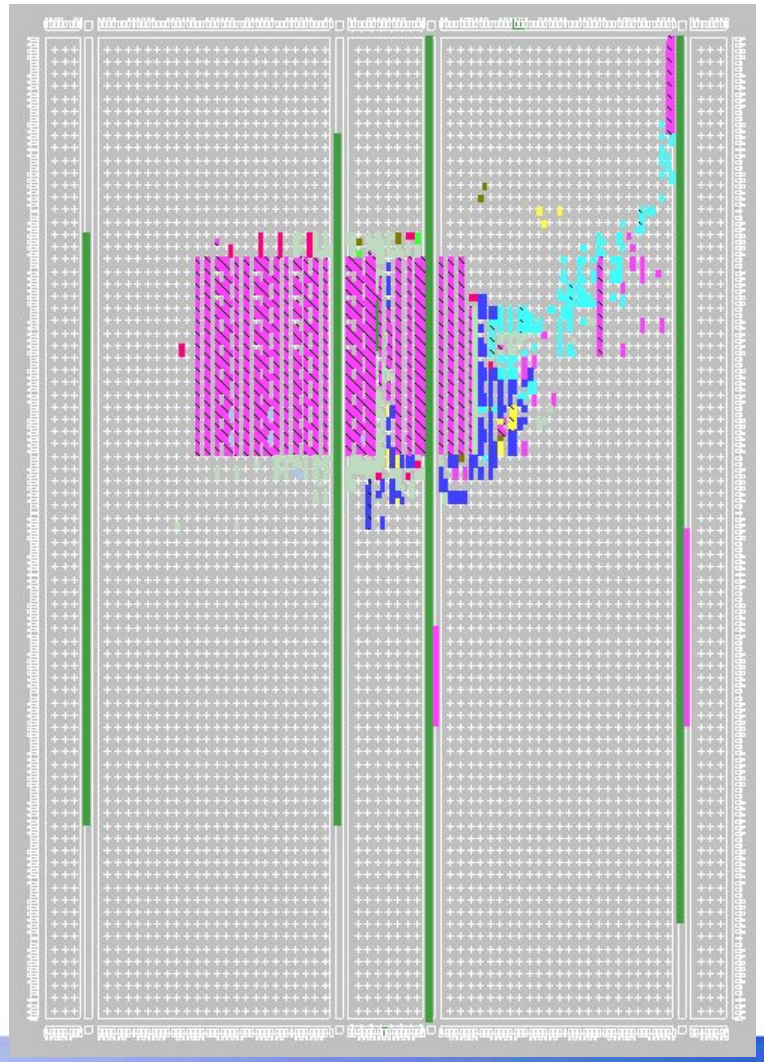
Hardware UART XC2V1000



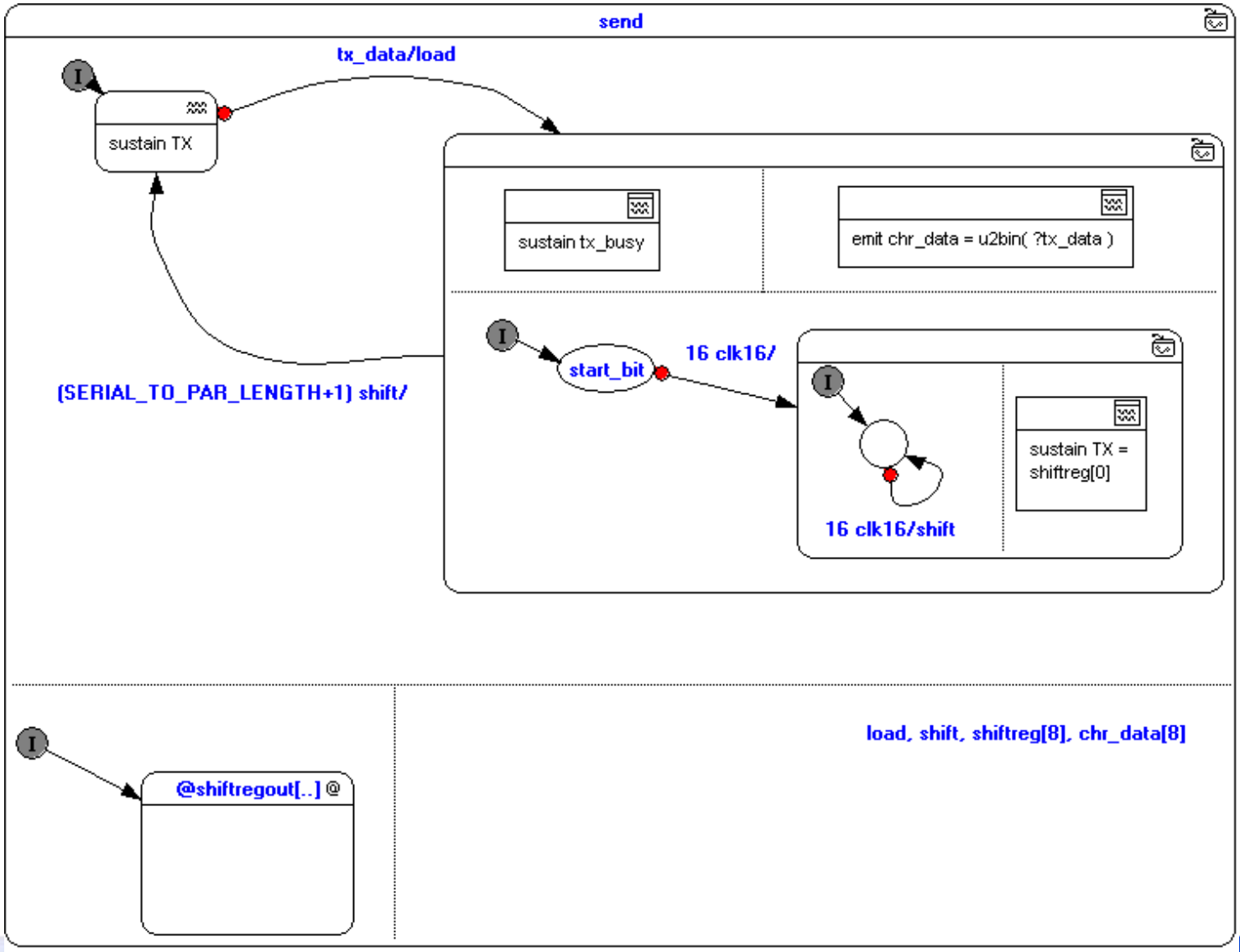
Direct use in SoC



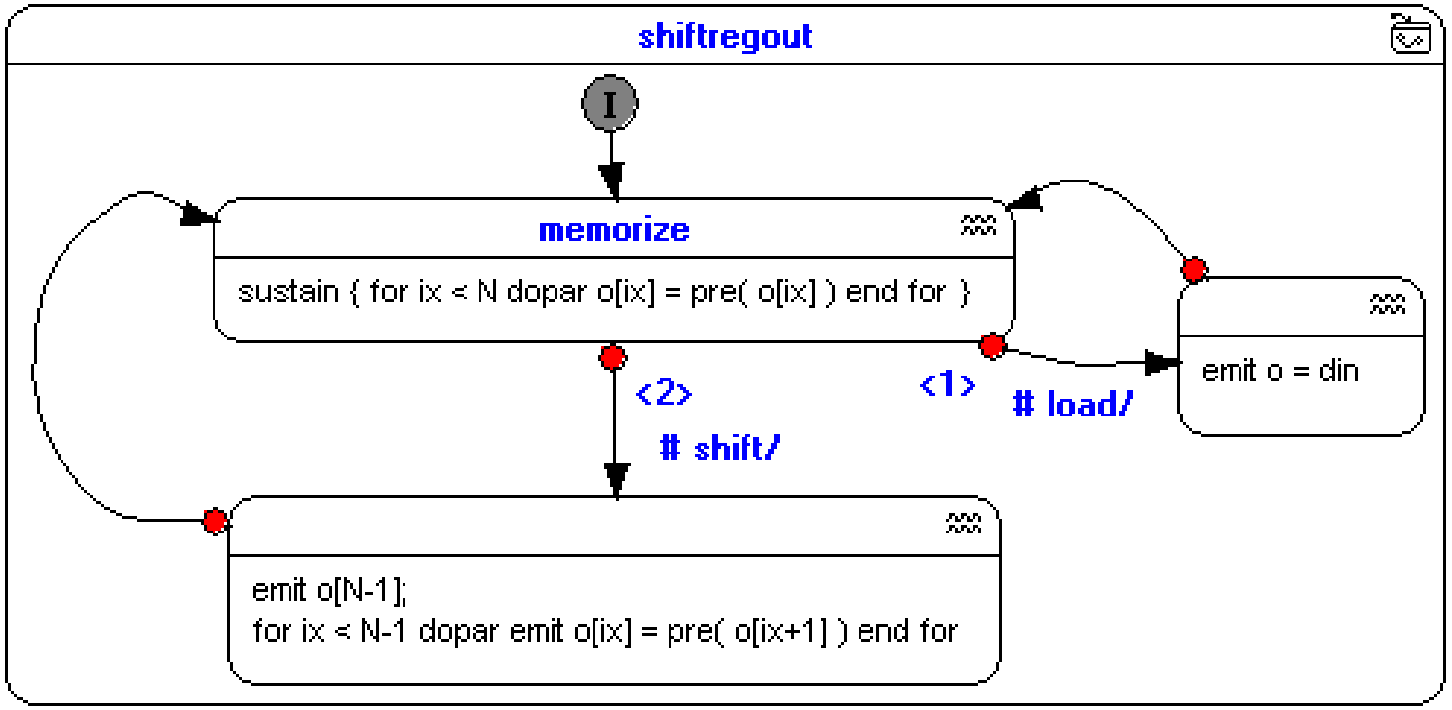
Soft UART MicroBlaze XC2V1000



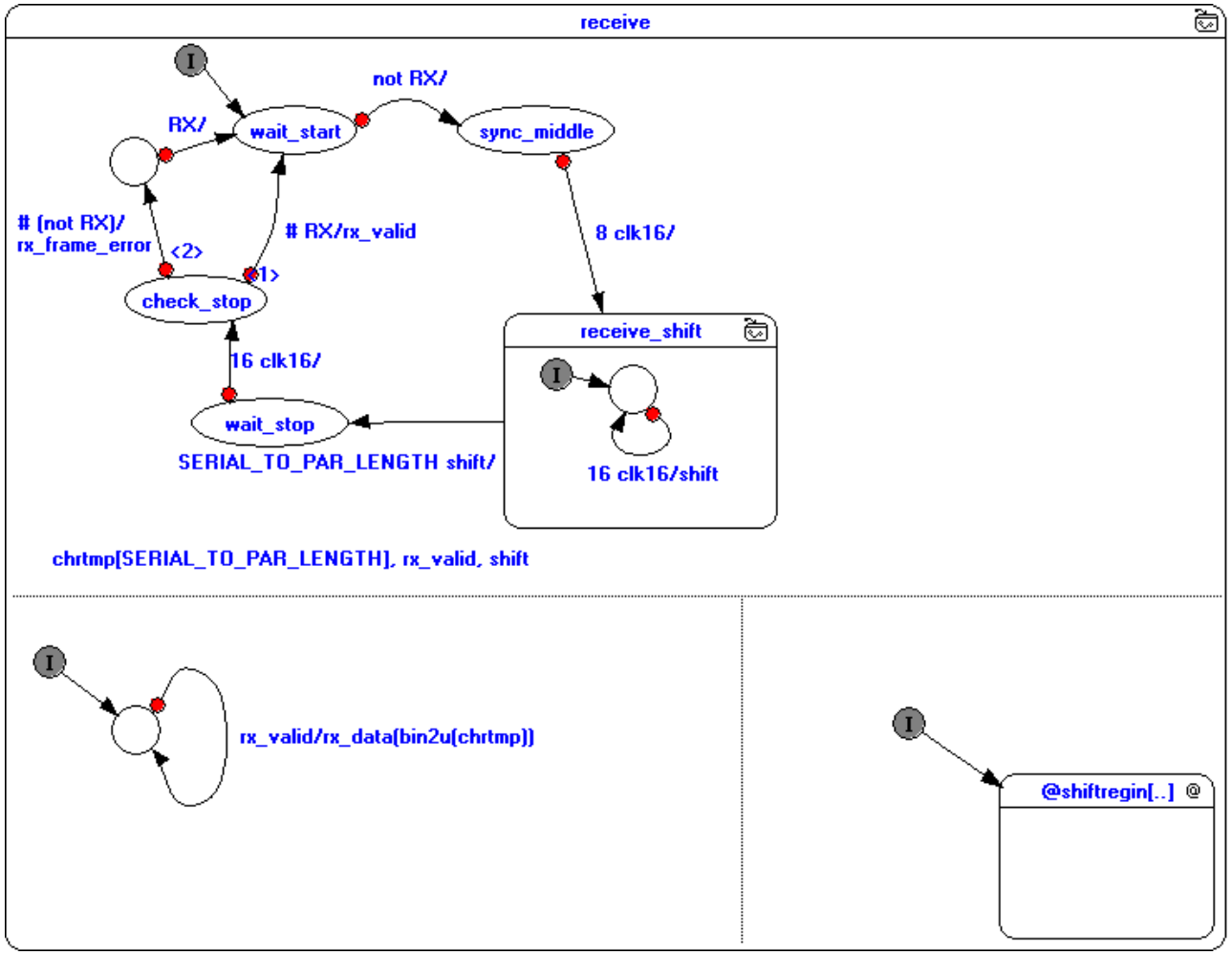
sender


send
tx_data/load
sustain TX
sustain tx_busy
emit chr_data = u2bin(?tx_data)
(SERIAL_TO_PAR_LENGTH+1) shift/
start_bit
16 clk16/
sustain TX =
shiftreg[0]
16 clk16/shift
@shiftregout[..] @
load, shift, shiftreg[8], chr_data[8]

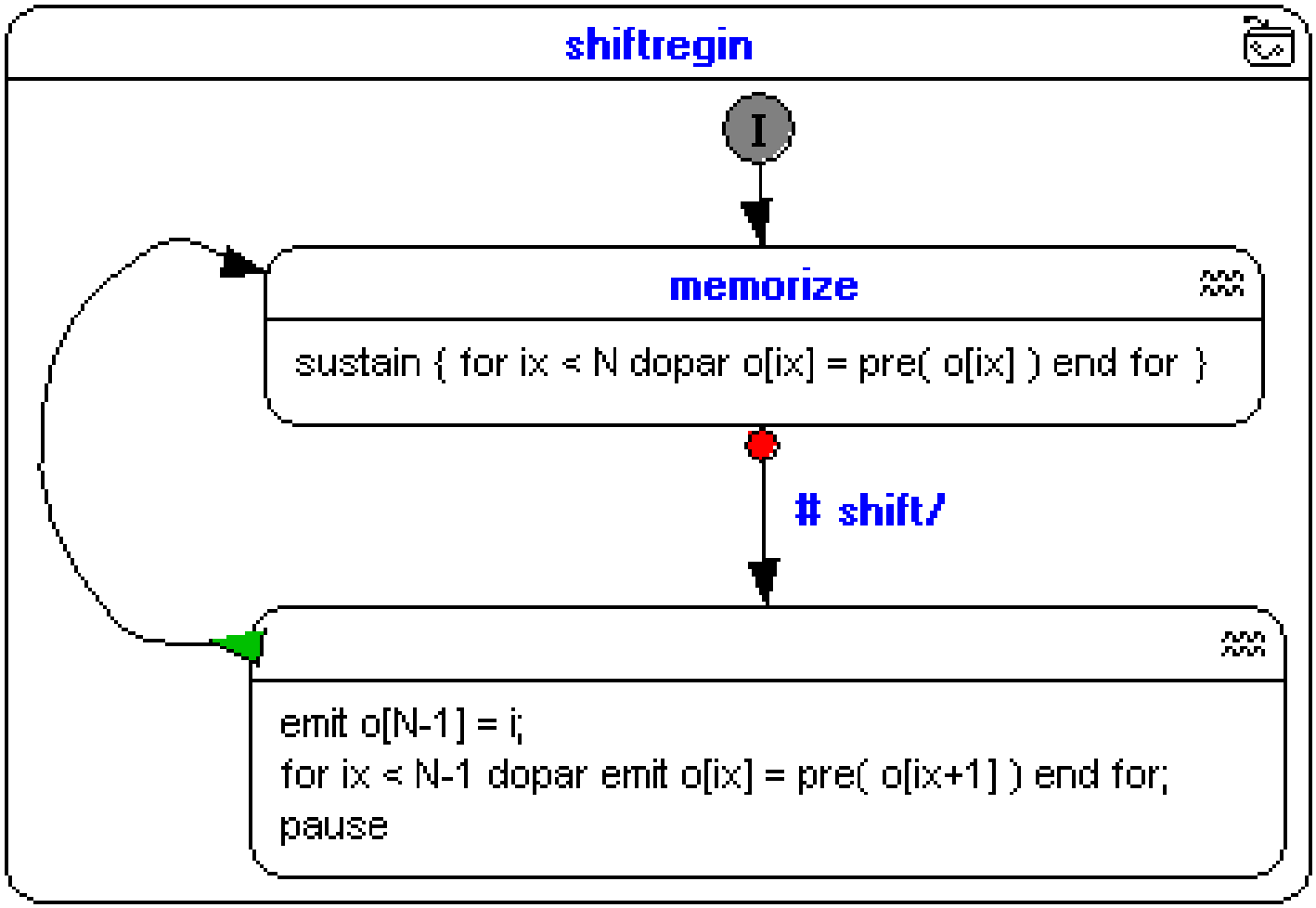
parallel to serial shift



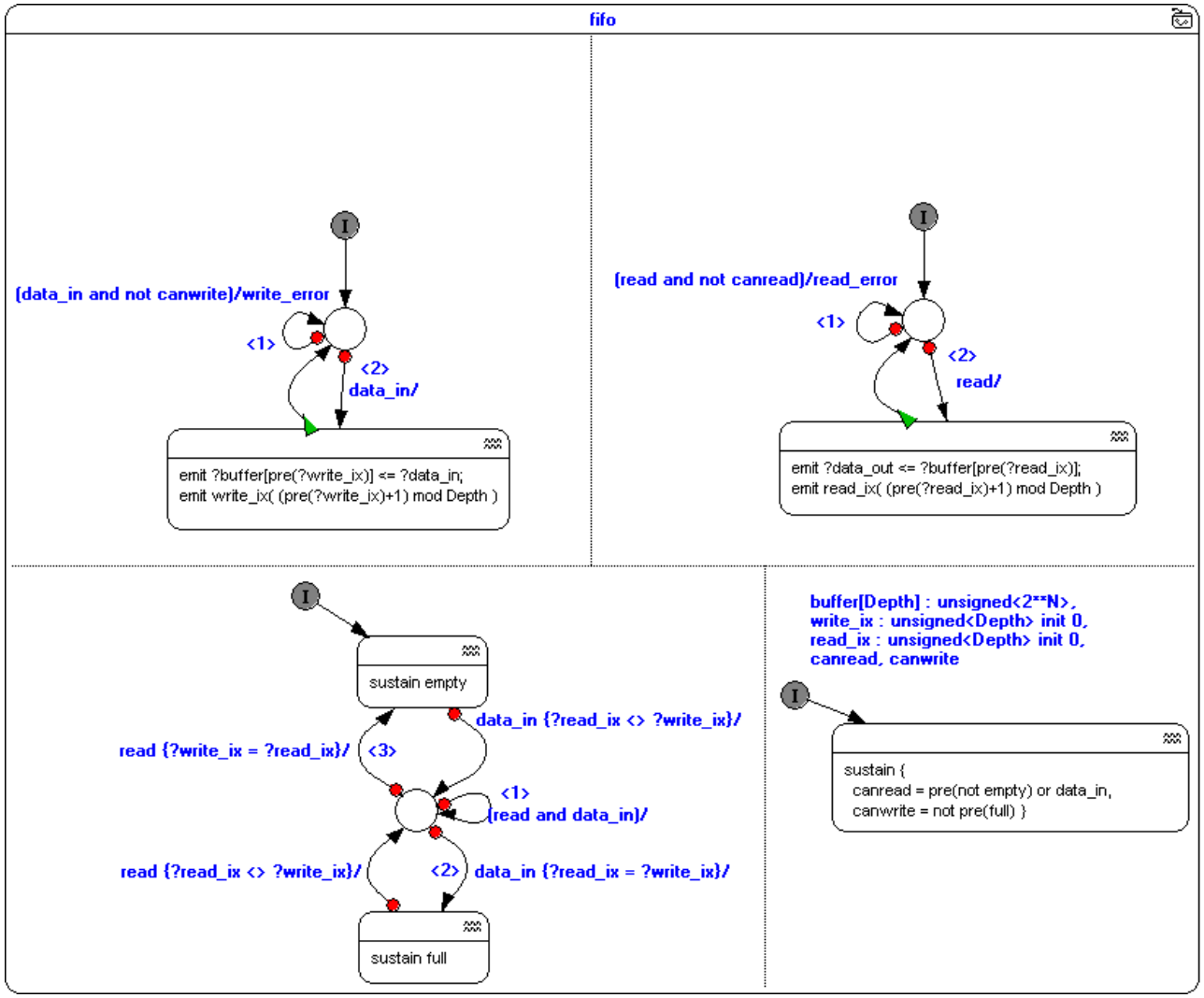
receive


receive
not RX/
RX/
wait_start
sync_middle
(not RX)/
rx_frame_error
<2>
RX/rx_valid
<1>
8 clk16/
check_stop
receive_shift
16 clk16/
wait_stop
SERIAL_TO_PAR_LENGTH shift/
16 clk16/shift
chrtmp[SERIAL_TO_PAR_LENGTH], rx_valid, shift
rx_valid/rx_data(bin2u(chrtmp))
@shiftregin[..] @

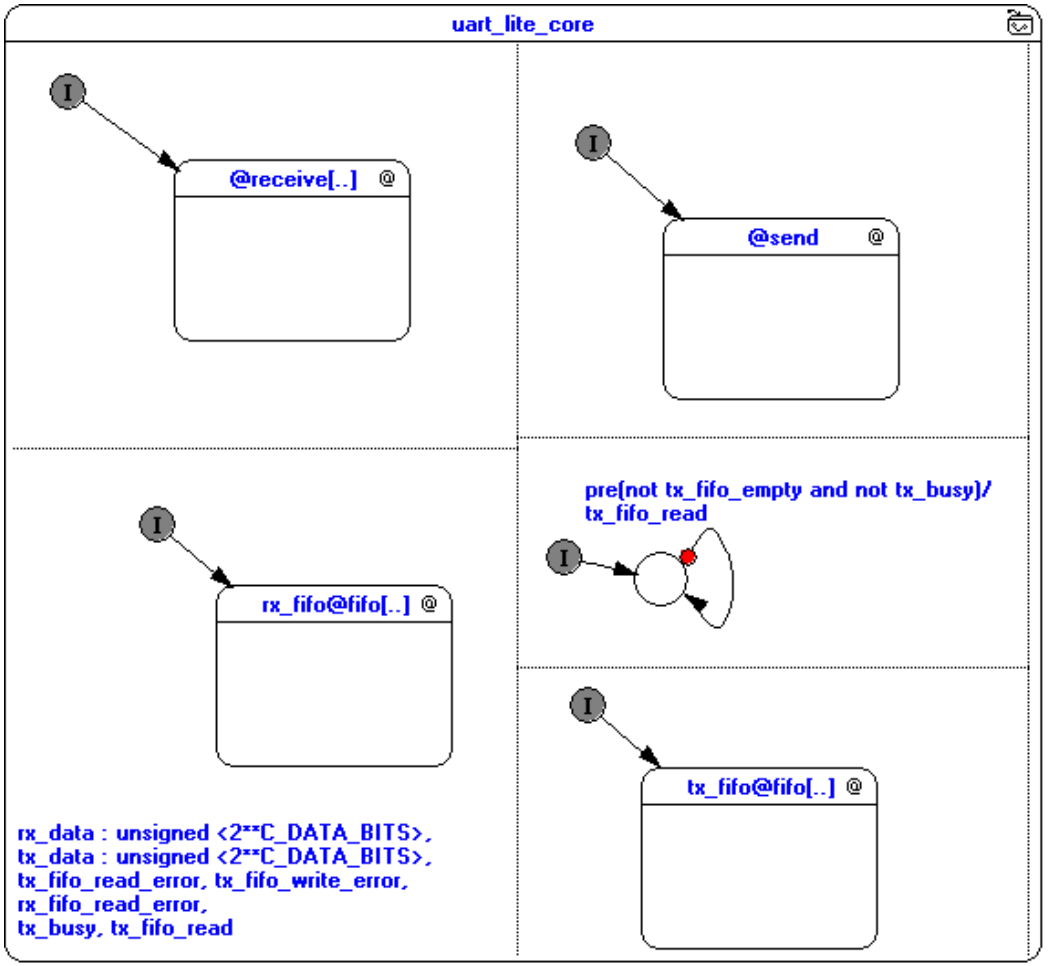
serial to parallel



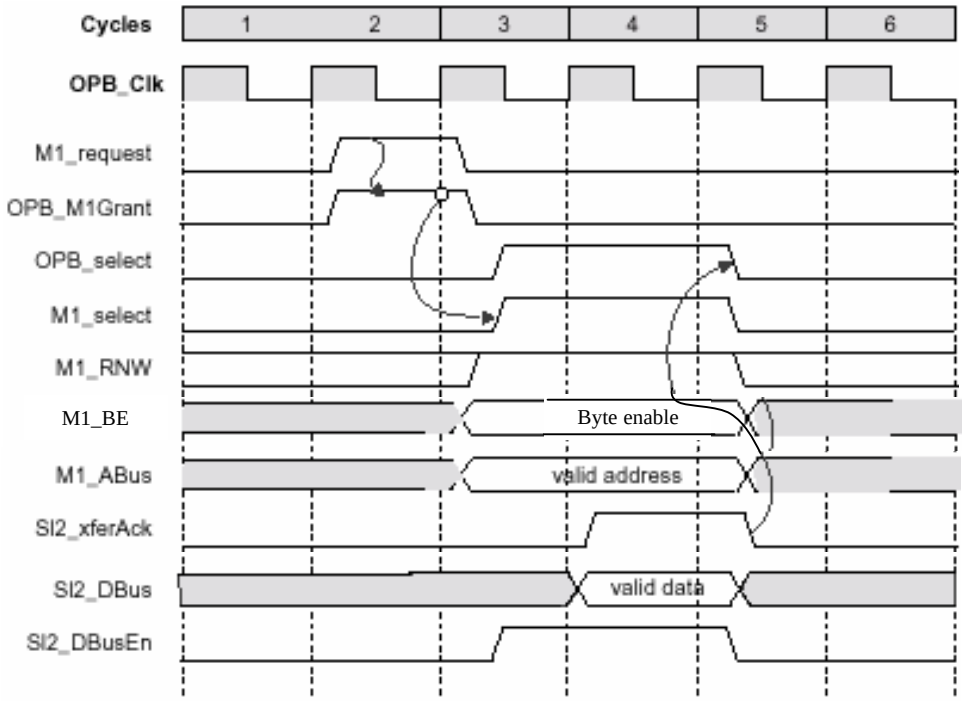
FIFO


fifo
(data_in and not canwrite)/write_error
<1>
<2>
data_in/
emit ?buffer[pre(?write_ix)] <= ?data_in;
emit write_ix((pre(?write_ix)+1) mod Depth)
(read and not canread)/read_error
<1>
<2>
read/
emit ?data_out <= ?buffer[pre(?read_ix)];
emit read_ix((pre(?read_ix)+1) mod Depth)
sustain empty
data_in {?read_ix <> ?write_ix}/
read {?write_ix = ?read_ix}/
<3>
<1>
(read and data_in)/
read {?read_ix <> ?write_ix}/
<2>
data_in {?read_ix = ?write_ix}/
sustain full
buffer[Depth] : unsigned<2**N>,
write_ix : unsigned<Depth> init 0,
read_ix : unsigned<Depth> init 0,
canread, canwrite
sustain {
canread = pre(not empty) or data_in,
canwrite = not pre(full) }

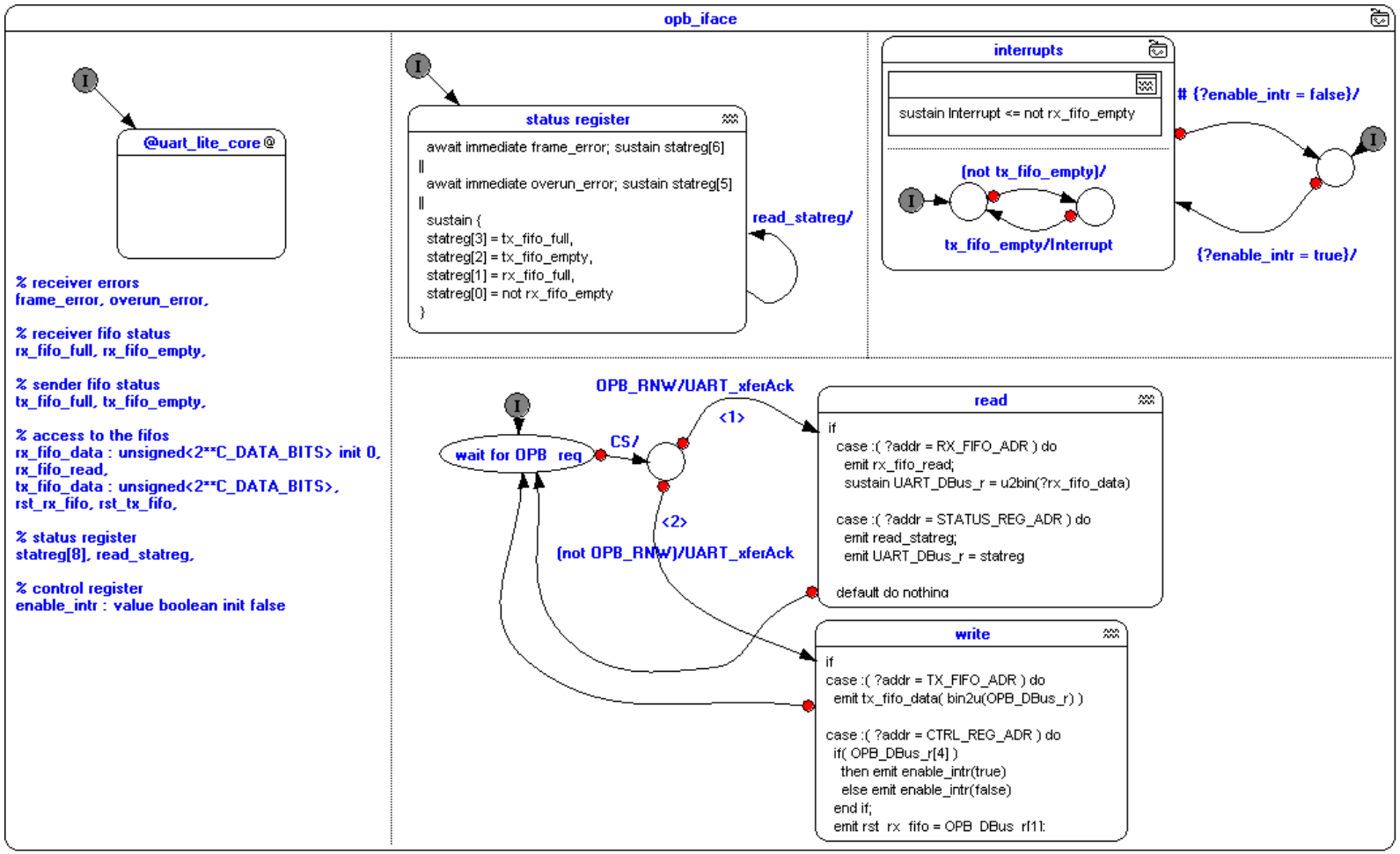
UART without bus interface



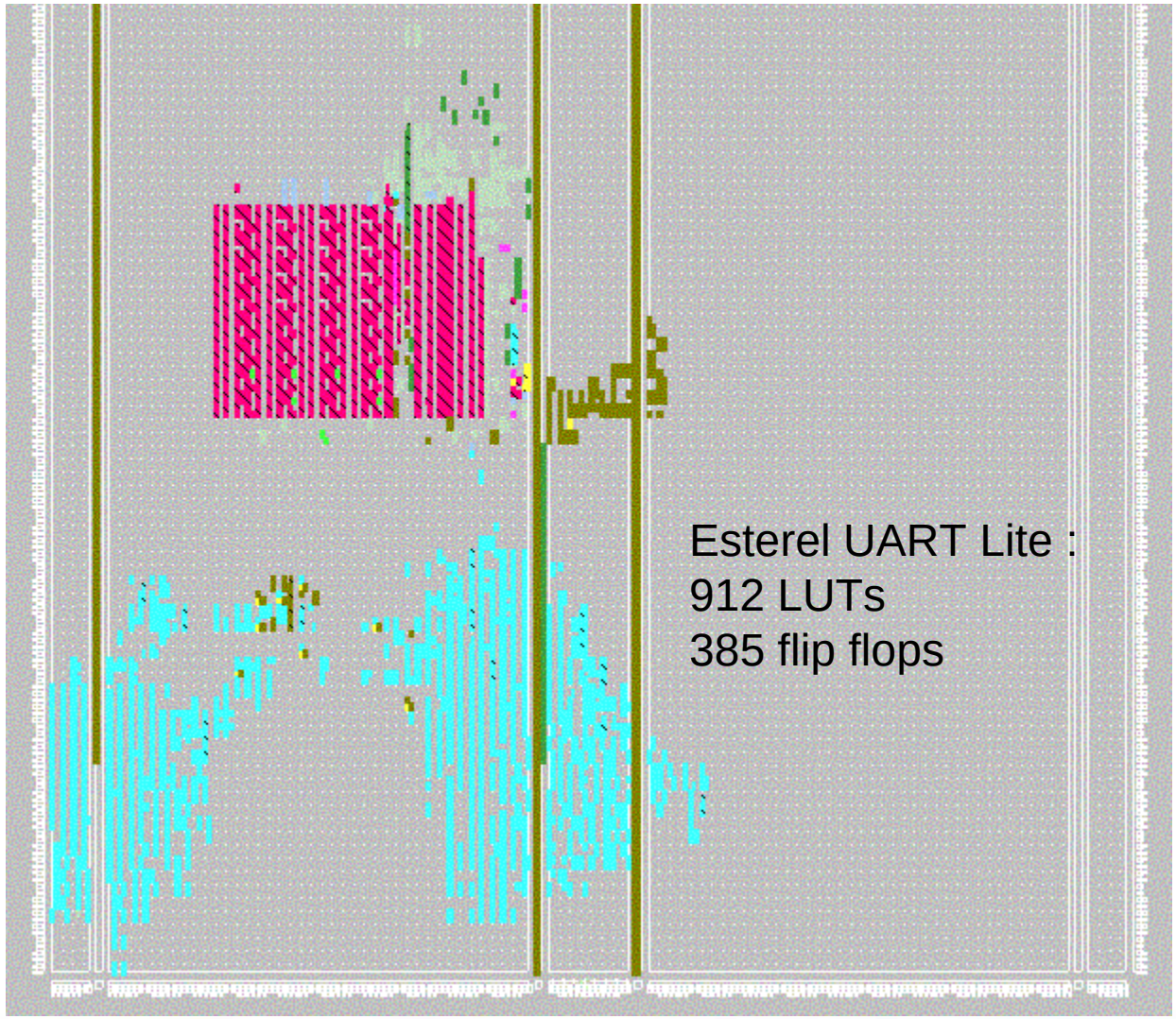
OPB Protocol



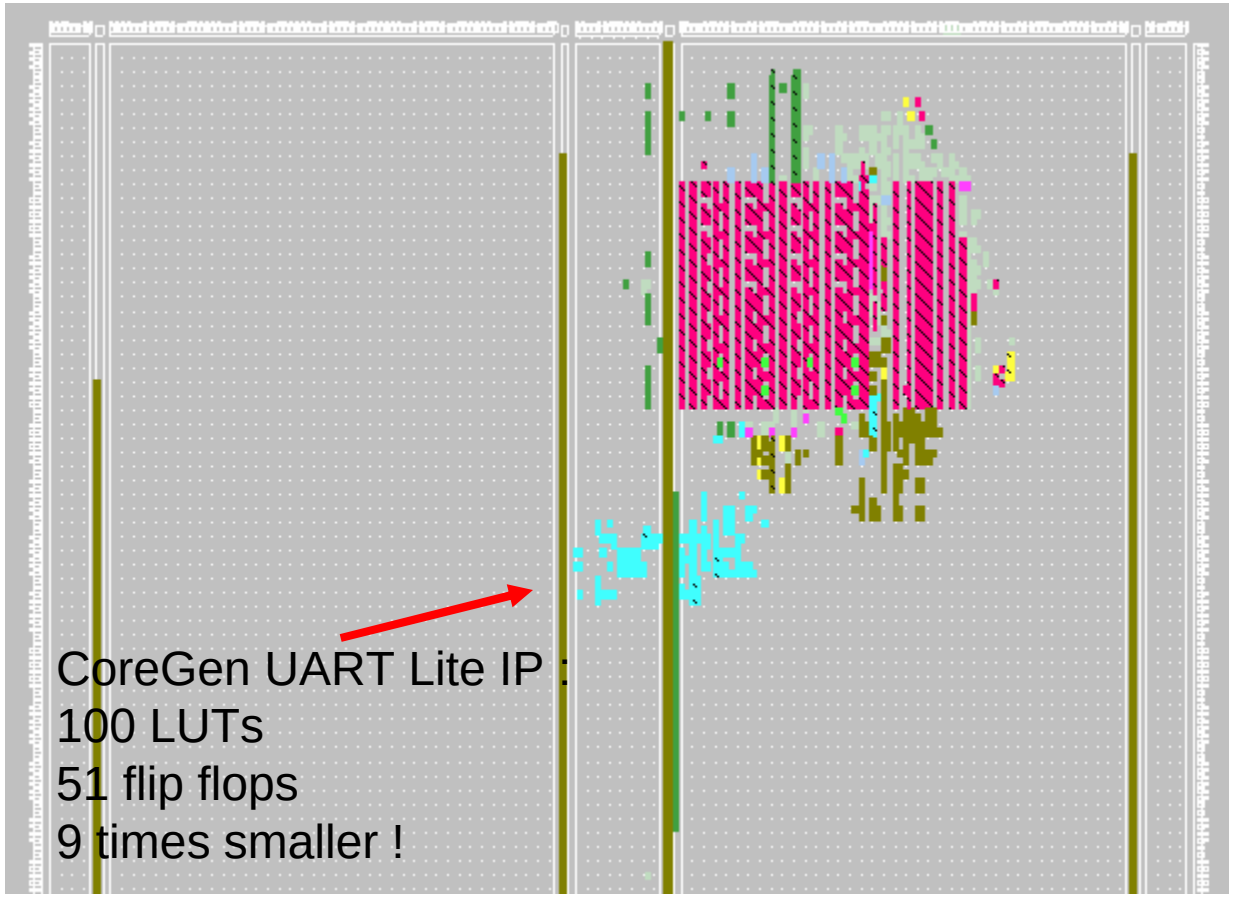
UART with OPB Interface



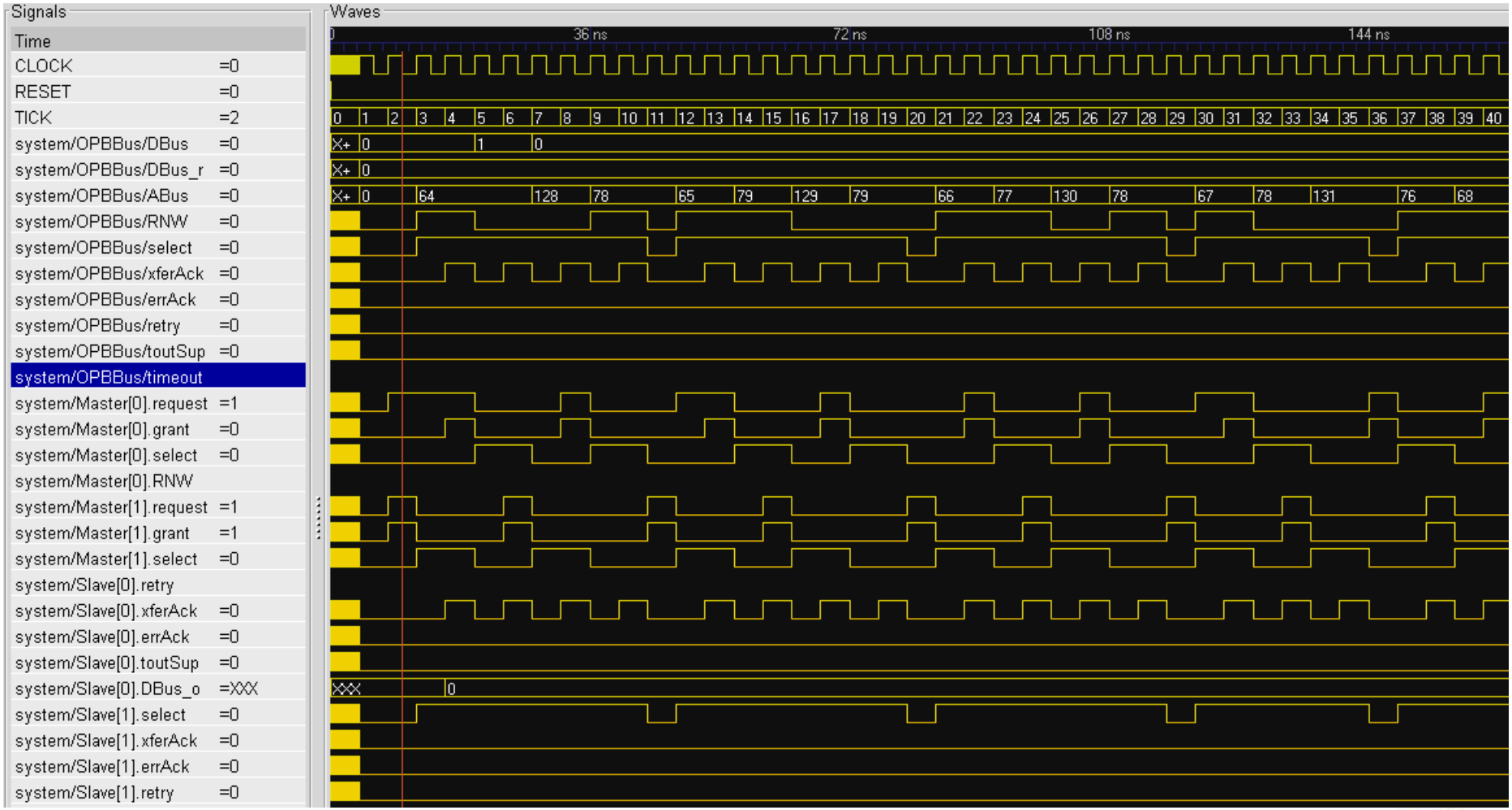
Generated circuit



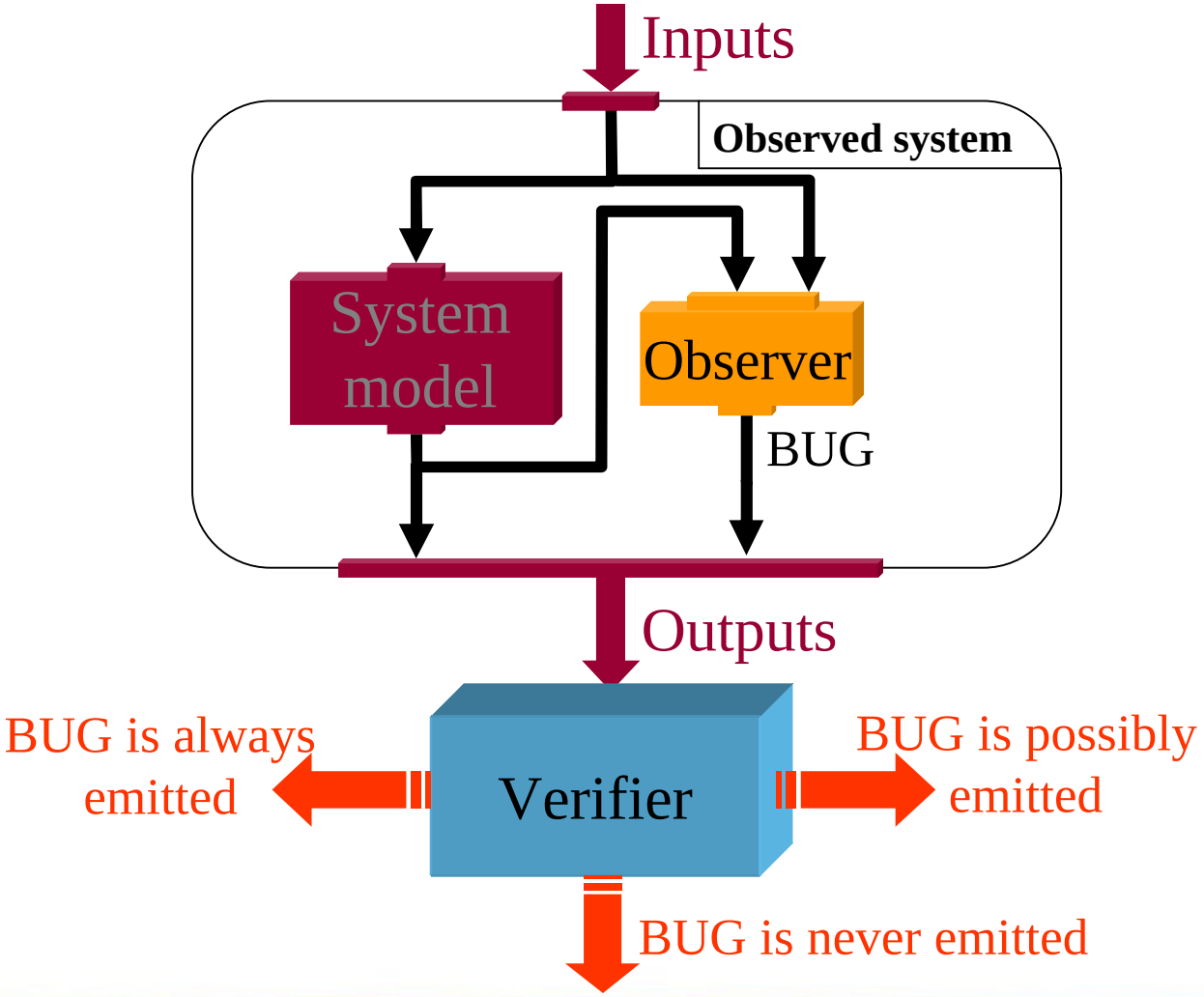
Comparison with Original CoreGen IP



Verification by simulation

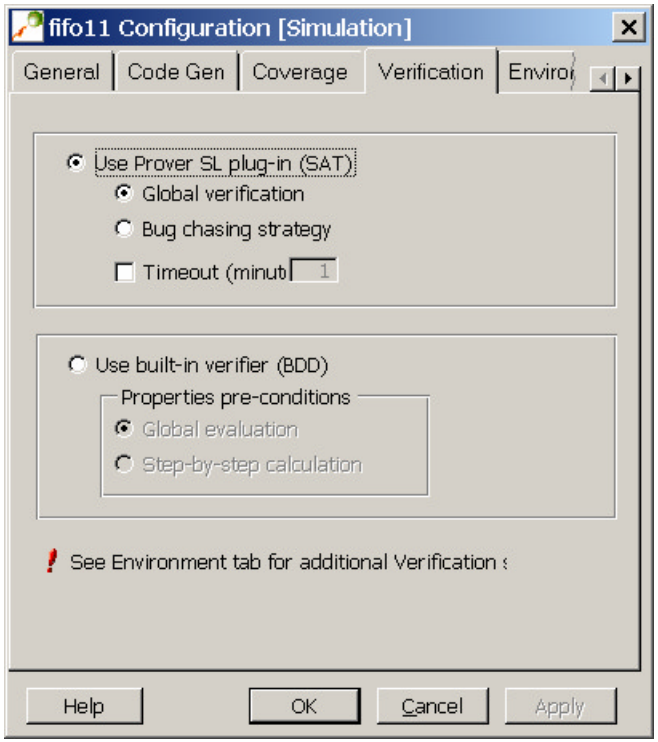


Verification with Observers



Verification engines

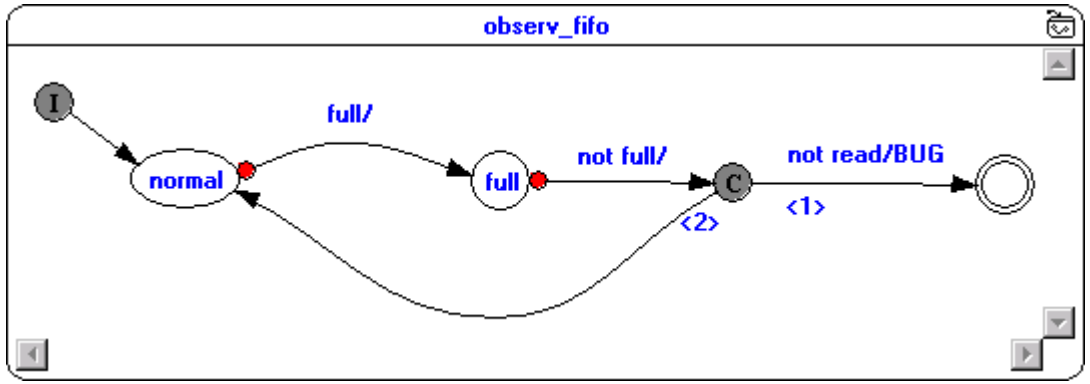
- 2 proof engines available inside Esterel Studio
  - Built-in verifier : TiGer
    - BDD technique
  - Prover Plug-in
    - SAT technique



Formal verification

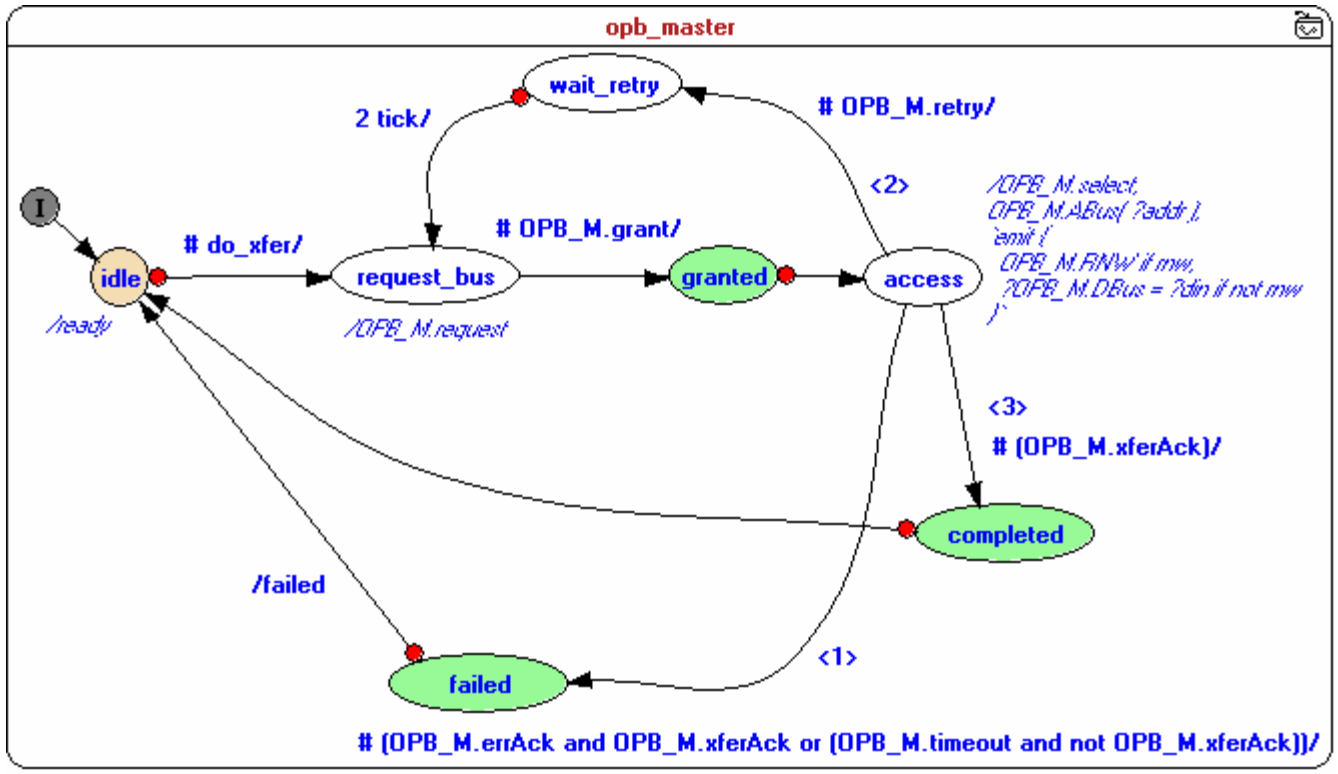
Of the FIFO :

proving that only a read access can make it exit the “full” state

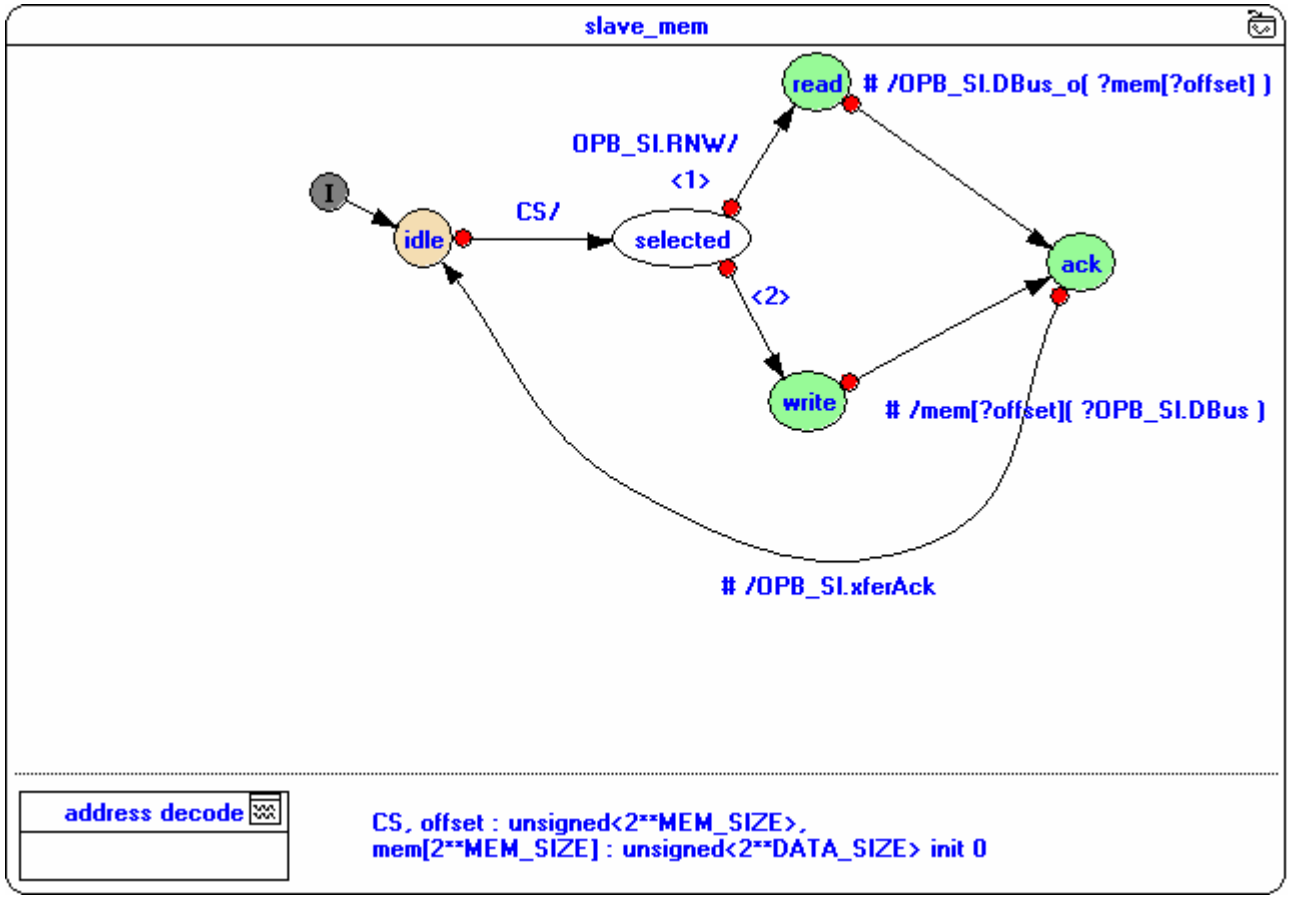


Proven in less than 2 seconds

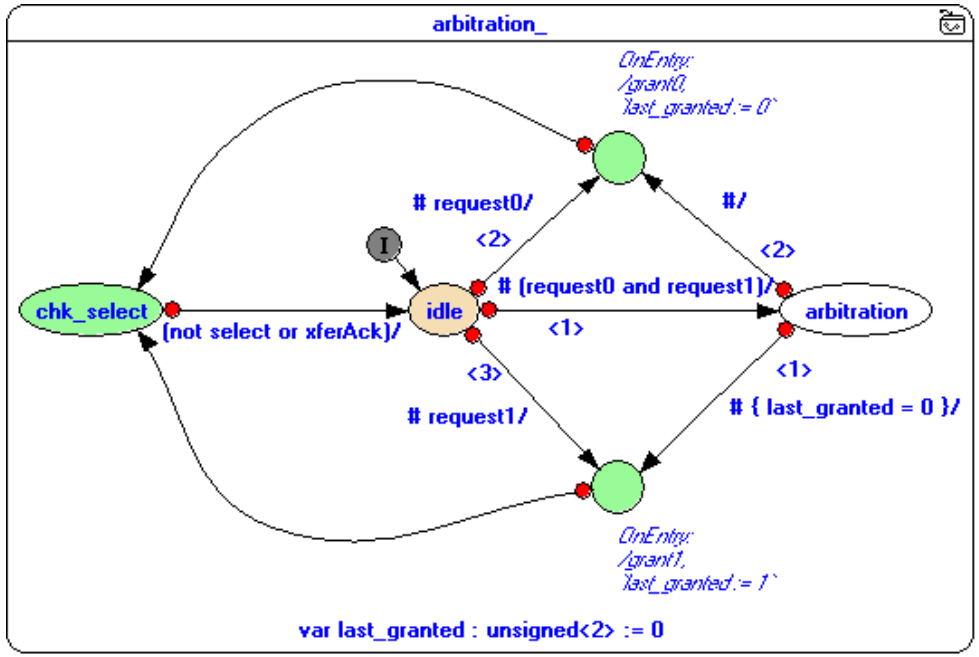
Specification of master behavior ...

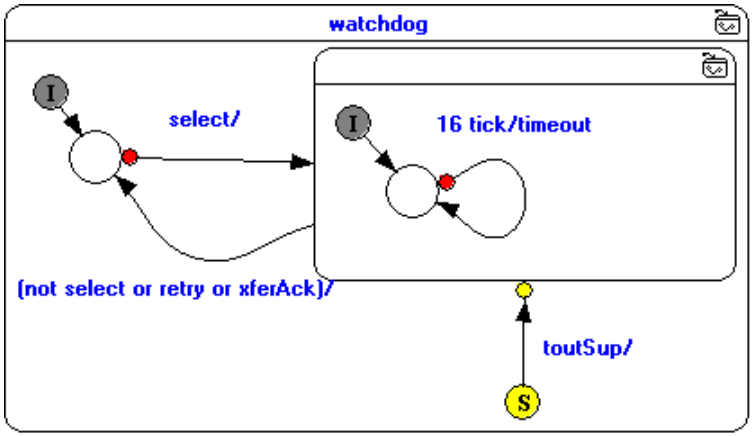


… slave


slave_mem
read
/OPB_SI.DBus_o(?mem[?offset])
OPB_SI.RNW/
<1>
I
CS/
idle
selected
ack
<2>
write
/mem[?offset](?OPB_SI.DBus)
/OPB_SI.xferAck
address decode
CS, offset : unsigned<2**MEM_SIZE>,
mem[2**MEM_SIZE] : unsigned<2**DATA_SIZE> init 0

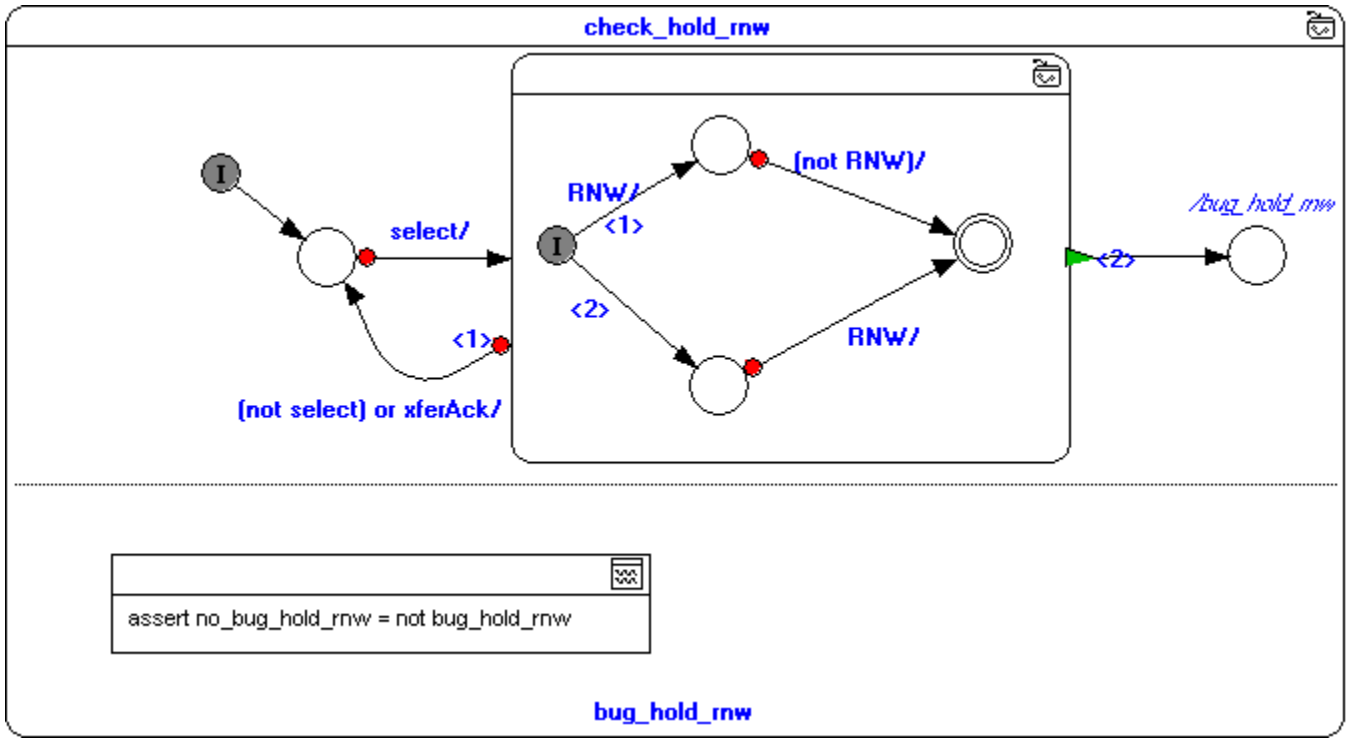
and arbiter


arbitration_
OnEntry:
/grant0;
'last_granted' := 0`
request0/
<2>
#/
<2>
I
chk_select
(not select or xferAck)/
idle
(request0 and request1)/
<1>
arbitration
<3>
request1/
<1>
{ last_granted = 0 }/
OnEntry:
/grant1;
'last_granted' := 1`
var last_granted : unsigned<2> := 0


watchdog
I
select/
I
16 tick/timeout
(not select or retry or xferAck)/
toutSup/
S

OPB Protocol violations

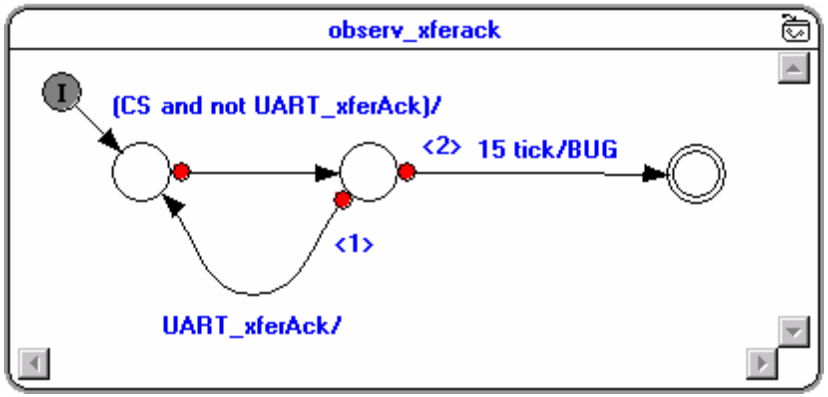
e.g. Checking that RNW doesn't change during a transaction :



Formal verification

Of the OPB slave interface :

proving that it won't cause bus timeouts



Results

Type	Name	Status
observer	observ_xferac_BUG	Always Absent

Verification engine used: Prover SL plug-in

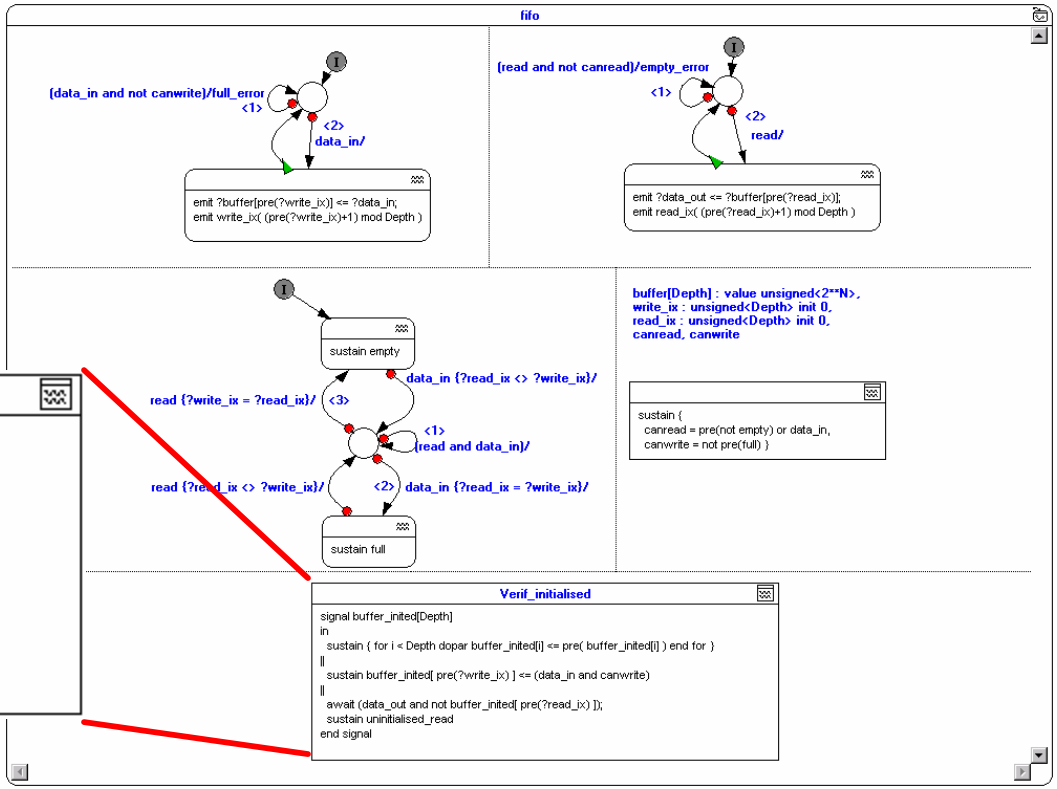
Proven in less than 2 seconds

Formal verification

Of the FIFO : proving that only initialized data is returned

Using an internal observer to access internal signals

No constraint on input signals



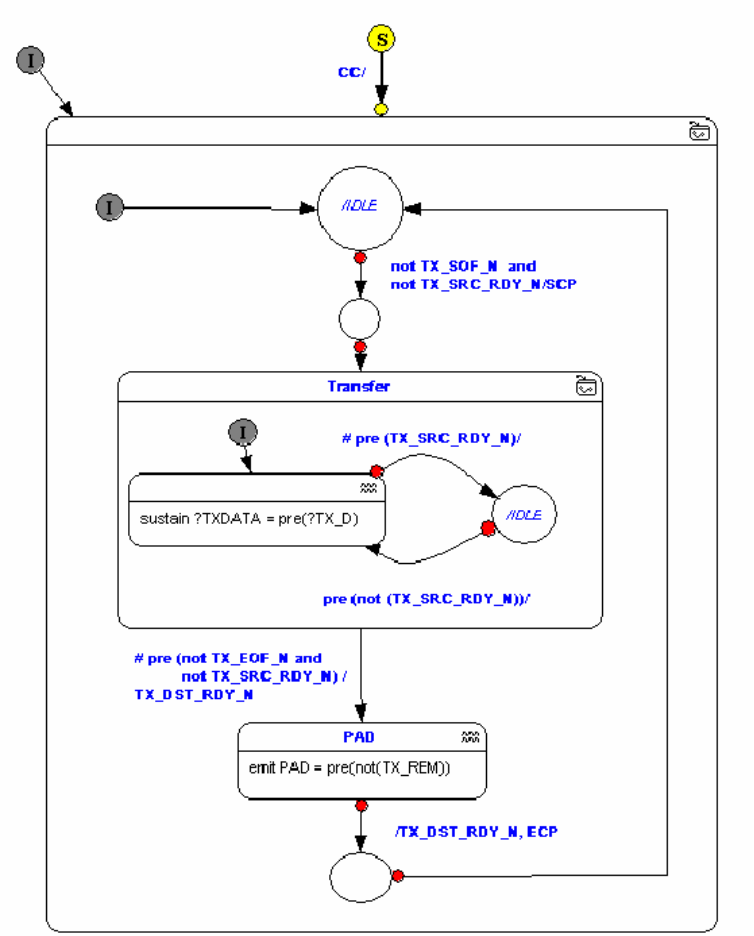
Verif_initialised

```
signal buffer_inited[Depth]
in
  sustain { for i < Depth dopar buffer_inited[i] <= pre( buffer_inited[i] ) end for }
||
  sustain buffer_inited[ pre(?write_ix) ] <= (data_in and canwrite)
||
  await (data_out and not buffer_inited[ pre(?read_ix) ]);
  sustain uninitialised_read
end signal
```

Proven in 30 seconds

Interactive Deadlock Demo

Other examples (LocalLink, Aurora, ...)



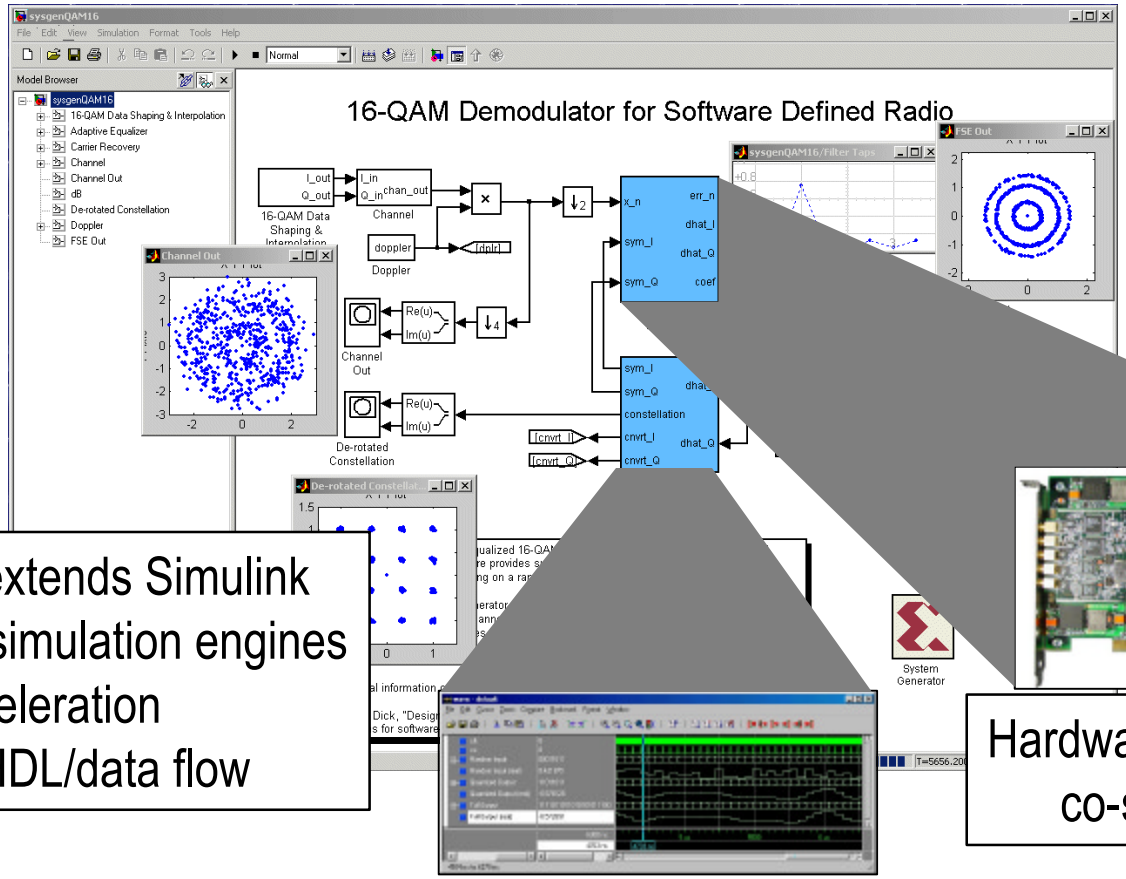
Positive Conclusions

- Control-based calculations can be implemented in hardware using a software style specification in Esterel ("computing without processors").
- Synchronous observers provide an additional verification technique to simulation, assertion languages (Sugar/OpenVERA etc.) and permits co-verification.
- Co-synthesis allows HW/SW trade-offs to be explored.
- VHDL/RTL provide poor interface between high systems and back-end tools.

Next Steps

- Currently working on:
  - Xilinx Link Layer protocol (LocalLink, Aurora).
  - TX portion of 10 gigabit ethernet MAC.
- Wire-speed high level processing of gigabit and 10-gigabit traffic.
- Language enhancements to better support HW design.
- Interface synthesis (a la CoWare)
- Control for System Generator

System Generator



System Generator extends Simulink to support external simulation engines

- Hardware acceleration
- Mixed-mode HDL/data flow

HDL co-simulation

Hardware in the loop co-simulation