Co-Design and Co-Verification using a Synchronous Language

Satnam Singh
Xilinx Research Labs
## Virtex-II PRO

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

- JPEG 2000 Software
- 2D DWT Hardware
- USB Camera Interface
- ZBT Memory Controller

PowerPC 405
Video Monitoring Application

- JPEG 2000 Software
- 2D DWT Hardware
- USB Camera Interface
- ZBT Memory Controller
- Ethernet Interface
- PowerPC 405
- Video Monitoring Software
Bus Based Reconfiguration

Diagram:
- Bus
- Device A
- Device B
- Reconfig Arbiter
Bus Based Reconfiguration
Bus Based Reconfiguration
Single Specification for Hardware and Software

HW/SW agnostic specification

VHDL, Verilog -> hardware implementation

C -> software implementation
Embedded Developer Kit
# CPU address space 0xFFFFE000 - 0xFFFFFFFF.
ADDRESS_BLOCK dramctlr
BUS_BLOCK [0xFFFFF000:0xFFFFFFFF]
xrefdes/dramctlr/bram0 [7:0] LOC=RAMB16_X0Y0;
xrefdes/dramctlr/bram1 [15:8] LOC=RAMB16_X1Y0;
xrefdes/dramctlr/bram2 [23:16] LOC=RAMB16_X2Y0;
...
END_BUS_BLOCK;
END_ADDRESS_BLOCK;
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.

```plaintext
loop
  abort
  { await Addr || await Data };
  emit Write(funcW(?Addr,?Data));
  await Latency tick
  when Replay
  x > 0 \( x := x - 1 \)
end loop
```
{ // equation style
    sustain{
        ReadEmpty = Read and pre(FifoIsEmpty)
        FifoIsEmpty = if (?Entries = 0) }
      ||
    // imperative style
    every DeltaEntries do
    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
Verification of project: demonstration

Use the right-click button to set the inputs and to select properties to verify in the tabs below.

Model outputs:
- G0
- G1
- G2
- H0
- H1
- H2
- P
- Q
- R
- S

Results:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Status</th>
<th>Counter example scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>H0</td>
<td>Never emitted</td>
<td></td>
</tr>
</tbody>
</table>

Verification engine used: Prover SL plug-in
Code generation

Esterel design

void uart_device_driver ()
{
    ....
}

uart.c

C -> software implementation

VHDL, Verilog -> hardware implementation

uart.c

C -> software implementation

uart_device_driver()
Hardware UART XC2V1000
Direct use in SoC
Soft UART MicroBlaze
XC2V1000
sender

send

- tx_data/load

- (SERIAL_TO_PAR_LENGTH+1) shift/

- start_bit

- 16 clk16/

- load, shift, shiftreg[8], chr_data[8]

- @shiftregout[..] @
parallel to serial shift

```plaintext
memorize

sustain { for ix < N dopar o[ix] = pre( o[ix] ) end for }

emit o = din

emit o[N-1];
for ix < N-1 dopar emit o[ix] = pre( o[ix+1] ) end for
```

# shift/

# load/
serial to parallel

```
shiftregin

I

memorize

sustain { for ix < N dopar o[ix] = pre( o[ix] ) end for }

# shift/

emit o[N-1] = i;
for ix < N-1 dopar emit o[ix] = pre( o[ix+1] ) end for;
pause
```
UART without bus interface
OPB Protocol
UART with OPB Interface
Generated circuit

Esterel UART Lite:
912 LUTs
385 flip flops
Comparison with Original CoreGen IP

CoreGen UART Lite IP:
- 100 LUTs
- 51 flip flops
- 9 times smaller!
Verification by simulation
Verification with Observers

- Inputs
  - Observed system
  - System model
  - Observer
    - BUG is possibly emitted
    - BUG is always emitted
  - BUG is never emitted

- Outputs
  - Verifier
    - BUG is always emitted
    - BUG is possibly emitted
    - BUG is never emitted
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 ...

\[ # \{ \text{OPB\_M\_retry} \} \]

\[ \text{idle} \rightarrow \text{request\_bus} \rightarrow \text{granted} \rightarrow \text{access} \]

\[ \text{wait\_retry} \rightarrow # \text{OPB\_M\_retry} \]

\[ \text{failed} \rightarrow \text{completed} \]

\[ # \{ \text{OPB\_M\_errAck and OPB\_M\_xferAck or (OPB\_M\_timeout and not OPB\_M\_xferAck)} \} \]
... slave
and arbiter
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

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

Proven in 30 seconds
Interactive Deadlock Demo
Other examples (LocalLink, Aurora, ...)

[Diagram of a state machine with transitions and conditions]

[Logo of Xilinx at the bottom right]
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

Hardware in the loop co-simulation