SystemVerilogCSP: Modeling Digital Asynchronous Circuits Using SystemVerilog Interfaces

Arash Saifhashemi¹
Peter A. Beerel¹,²

¹Ming Hsieh Dept. of Electrical Engineering, University of Southern California
²Fulcrum Microsystems, Calabasas CA, 91302.

CPA 2011
University of Limerick, Ireland
• Introduction
  • Why asynchronous circuit design?
  • Hardware description languages

• SystemVerilog Abstract Communication—Basic Features
  • Channels - Send, Receive
  • Channel status and mixed-level simulation

• SystemVerilog Abstract Communication—Extended Features
  • Peek and Probe
  • Split and synchronized communication
  • One-to-many and one-to-any channels

• Results and Conclusion
Why Asynchronous Circuit Design?

- Systems on chip: chips are becoming distributed systems
  - Communication-dominated
  - No globally synchronous clock
- Asynchronous alternative
  - Local handshaking: CSP-like communication
- Benefits
  - Higher speed
    - Shorter globally critical paths
  - Lower power consumption
    - Remove power-hungry global clock
    - Modules active only when necessary
  - Robustness to variations
    - Process, voltage, and temperature

Pictures: [Benini’06]
Asynchronous Circuit Design - Today

Applications
- Ethernet Switches (Fulcrum Microsystems)
- Ultra high-speed FPGAs (Achronix)
- Multi-core network on Chip
- Ultra low-power chip design

Basic challenges
- Lack of CAD tools to automate designs

Proteus design flow (USC)
- Leverage off of available synchronous CAD tools
- Starting at a high-level specification written in SystemVerilogCSP

Flowchart:
- Proprietary CSP
  - SystemVerilog
  - Standard Synthesis
  - Asynchronization and Optimization
  - Standard Physical Design
  - Final Layout

CAD Tools Commercially Available
Hardware Description Languages

• Desirable features of an HDL
  • Concurrency e.g.: A=B || (C=D ; E=F)
  • Timing e.g.: A=B after 5ns
  • Support for various levels of abstraction
  • Support by commercial CAD tools
  • Support for both synchronous & asynchronous

• Communication abstraction
  • Ease of design
  • Design usability: protocols evolve and change
  • Architecture evaluation before implementation
  • Ease of adoption by synchronous designers

• CSP as a basis for a hardware description language
  • Suitable for modeling abstract communication
  • Lacks some desirable features
Previous Work

• **New Language inspired by CSP**
  • Have limited CAD tool support - LARD [Edwards et al], Tangram [Berkel et al], CHP [Martin]

• **Software languages**
  • No inherent support for timing, limited CAD tool support - JCSP [Welch et al]

• **VHDL**
  • Fine grained concurrency is cumbersome [Frankild et al, Renaudin et al, Myers et al]

• **VerilogCSP**
  • Verilog Programming Language Interface: very slow; cannot handle multi-channel modules [Saiflashemi et al]
  • Verilog macros are cumbersome and do not support extensions

• **SystemVerilog (Superset of Verilog)**
  • Initial implementations promising but do not address extensions [Tiempo]
• Abstract communication between processes

\[ \text{SENDER} = (\text{mid}!v \rightarrow \text{SENDER}) \]
\[ \text{RECEIVER} = (\text{mid}?x \rightarrow \text{RECEIVER}) \]

• No notion of **hardware implementation details**

• Semantics based on **events** on **channels** between independent **processes** [Hoare’ 04]
Introduction
  • Why asynchronous circuit design?
  • Hardware description languages

SystemVerilog Abstract Communication—Basic Features
  • Channels - Send, Receive
  • Channel status and mixed-level simulation

SystemVerilog Abstract Communication—Advanced Features
  • Peek and Probe
  • Split and synchronized communication
  • One-to-many and one-to-any channels

Results and Conclusion
Abstract SystemVerilog Channels

• Our approach
  • Use SystemVerilog interface to abstract **channel wires** as well as **Send/Receive tasks**

```
module Sender (interface R);
  parameter WIDTH = 8;
  logic [WIDTH-1:0] v;
  always begin
    v=${random()}%2**(WIDTH-1);
    R.Send(v);
    #10ns;
  end
endmodule
```

```
module Receiver (interface L);
  parameter WIDTH = 8;
  logic [WIDTH-1:0] x;
  always begin
    L.Receive(x);
    #15ns;
  end
endmodule
```
Behind The Scenes: Channel Interface

- Channel details encapsulated within an “Interface”
- Implementation details (below) hidden from user
  - Greatly simplifies debugging and evaluation of the design

```vhdl
typedef enum {idle, r_pend, s_pend} ChannelStatus;
typedef enum {P2PhaseBD, P4PhaseBD} ChannelProtocol;

interface Channel;
  parameter WIDTH = 8;
  parameter ChannelProtocol hsProtocol = P2PhaseBD;
  ChannelStatus status = idle; // Status of a channel
  logic req=0, ack=0; // Handshaking signals
  logic hsPhase=1; // Used in two-phase handshaking
  logic [WIDTH-1:0] data; // Data being communicated
endinterface: Channel
```
Send/Receive tasks are analogous to CSP’s ! (output) and ? (input).

Semantics are based on synchronization of concurrent processes using SystemVerilog’s notion of update and evaluation events.
Viewing Channel Status

- Enumerated types make viewing channel status inherent to all standard SystemVerilog simulators.
- The designer can monitor if and what processes are engaged in the communication over time.

Status of channels as a waveform.
Supports Mixed-Levels of Abstraction

- **Completed blocks** can be simulated with others **still at behavioral level**

```verilog
module mp_fb_csp (interface L, interface R);
  logic data;
  always begin
    L.Receive(data);
    R.Send(data);
  end
endmodule
```

```verilog
module mp_fb_gate (interface L, interface R);
  celement ce(L.req, pd_bar, c);
  not inv (pd_bar, pd);
  cap_pass cp (c, L.ack, R.ack, pd, L.data, R.data);
endmodule
```
Supports Design Verification

- Co-simulation: Implemented circuit vs. original circuit
- It is important to use the same Testbench
  - Sometimes very complicated
  - Verifies correct implementation
- No need for Shims [Saifhashemi’05]
Outline

• Introduction
  • Why asynchronous circuit design?
  • Hardware description languages
• SystemVerilog Abstract Communication— Basic Features
  • Channels - Send, Receive
  • Channel status and mixed-level simulation
• SystemVerilog Abstract Communication— Advanced Features
  • Peek and Probe
  • Split and synchronized communication
  • One-to-many and one-to-any channels
• Results and Conclusion
## Peek and Probe

### Peek
- Sample data without committing to communication

### Probe
- Is the channel idle?
- Usually used for arbitration

```verilog
task Peek (output logic[WIDTH-1:0] d);
    wait (status == s_pend);
    d = data;
endtask
```

```verilog
wait(ch0.status!=idle && ch1.status!= idle);
winner = Arbitrate (ch0.status, ch1.status);
if(winner == 0)
    ch0.Receive(d);
if(winner ==1)
    ch1.Receive(d);
```
Split Communication

• Handshaking of different channels might be **interleaved in implementation**
• Modeling interleaved behavior at high level is important for **early system evaluation**

```verilog
module buf (interface L, interface R);
  logic data;
  always begin
    L.Receive(data);
    R.Send(data);
  end
endmodule

module buf_split (interface L, interface R);
  logic data;
  always begin
    L.SplitReceive(data, 1);
    R.Send(data);
    L.SplitReceive(data, 2);
  end
endmodule
```
Synchronized Communications

- Sometimes implementation forces correlation of communication on multiple channels
- Synchronized start
- Synchronized finish
- Early performance evaluation of system requires modeling such behavior

Concurrent body starts

Always begin
fork
A.Receive(a, 1);
B.Receive(b, 1);
join
fork
A.Receive(a, 2);
B.Receive(b, 2);
join
sum = a + b ;
SUM.Send(sum);
end

Acts like a barrier
One-To-Many Channels

- One sender to multiple receivers
  - Option 1: Use a copy block
    - Makes design **cumbersome**
  - Option 2: Shared channels [JCSP, Welch et. al]
    - Sender and receiver send and receive **as if the channel is a normal one-to-one channel**
    - Top level module specifies the channel is broadcast

- Shared channels are closer to hardware implementation
  - A shared data bus between sender and receivers
  - Separate *req* and *ack* signals for receiving processes.
One-To-Any Channel

- One sender to multiple receiver - JCSP [Welch et. al]
- Only one of the receiver participates in communication

```verilog
task Receive(output logic[WIDTH-1:0] d);
  status = r_pend;
  wait (req == hsPhase);
  if (ONE2ANY)
    req = 'z; // Inhibits
    // other receivers
  d = data;
  ack = hsPhase;
  status = idle;
endtask

always begin : main
  wait (L.status != idle);
  randValue = {$random()} % 3 ;
  if (randValue ==1)
    L.Receive(x);
  else
    begin
      #0;
      disable main;
    end
end
```
• Introduction
  • Why asynchronous circuit design?
  • Hardware description languages
  • CSP Communication Channels
• SystemVerilog Abstract Communication– Basic Features
  • Channels - Send, Receive
  • Channel status and mixed-level simulation
• SystemVerilog Abstract Communication- Advanced Features
  • Peek and Probe
  • Split and synchronized communication
  • One-to-many and one-to-any channels
• Results and Conclusion
Results – Simulation Run-Times

• Comparison to VerilogCSP [Saifhashemi’05]
  • Simulation time of a linear pipeline with depth of 10
  • Platform: Sun UltraSPARC, Modelsim SE 6.6 simulator
  • 12%-20% improvement

<table>
<thead>
<tr>
<th>Number of data items</th>
<th>100K</th>
<th>200K</th>
<th>300K</th>
<th>400K</th>
<th>500K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation time in</td>
<td>45.14</td>
<td>76.38</td>
<td>107.60</td>
<td>139.57</td>
<td>170.62</td>
</tr>
<tr>
<td>Seconds (VerilogCSP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation time in</td>
<td>40.12</td>
<td>65</td>
<td>89.70</td>
<td>115.52</td>
<td>141.99</td>
</tr>
<tr>
<td>Seconds (SystemVerilogCSP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio</td>
<td>1.12</td>
<td>1.17</td>
<td>1.19</td>
<td>1.20</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Conclusions

• CSP-like communication and extensions can be modeled using SystemVerilog interfaces

• Features and advantages
  • Ease of design: abstract communication, channel status
  • Mixed asynchronous and synchronous designs can be modeled in same language and simulation environment
  • Extensions: more accurate modeling of implemented hardware
  • Make adoption of asynchronous technology easier

• Currently being used to teach the course EE-552 Asynchronous VLSI at the University of Southern California

• Future work
  • Automated synthesis from SystemVerilogCSP
Supports Design Verification

- Testing DUT:
  - Initially, modeled in SystemVerilogCSP
  - Later implemented in gates
- It is important that Testbench does change
  - Sometimes very complicated
  - Communicates with other blocks
  - Verifies correct implementation
- No need for Shims [Saifhashemi’05]