Definition

shim \ˈshim\  n

1 : a thin often tapered piece of material (as wood, metal, or stone) used to fill in space between things (as for support, leveling, or adjustment of fit).

2 : Software/Hardware Integration Medium, a language for describing hardware/software interfaces
Motivation

Combine two well-known semantic models in a single language:

- Single-threaded software
- Synchronous RTL hardware
Conceptually: Memory-mapped I/O
Actual FPGA Implementation

Implemented on a Xilinx Spartan-II FPGA using the Microblaze processor and OPB bus.
module timer {
  shared uint:32 counter;  // Visible to HW and SW

  hw void count() {  // Hardware process
    counter = counter + 1;
  }

  out void reset_timer() {  // Software function
    counter = 0;
  }

  out uint get_time() {  // Software function
    return counter;
  }
}
#ifndef _TIMER_H
#define _TIMER_H
extern void reset_timer(void);
extern unsigned int get_time(void);
#endif /* _TIMER_H */
```c
#include "timer.h"
#include "xio.h"
#define IO_BASE 0xfeff0200 /* I/O address */
#define counter (IO_BASE + 0x0)

void reset_timer() {
    XIo_Out32(counter, 0); /* Write 32 bits */
}

unsigned int get_time() {
    return XIo_In32(counter); /* Read 32 bits */
}
```
signal counter : UNSIGNED(31 downto 0); -- shared variable

count : process(Clk)
begin
  if Clk'event and Clk = '1' then
    counter <= counter + 1; -- body of one process
    if cs1 = '1' and RNW = '0' then
      if offset = 0 then
        counter <= DBus; -- software takes write precedence
      end if; end if; end if;
  end if;
end process count;

read_shared_variables : process(Clk)
begin
  if Clk'event and Clk = '1' then
    if cs1 = '1' and RNW = '1' then -- "chip select"
      DBus_out <= read_data;
    else DBus_out <= "0"; end if;
    if offset = 0 then
      read_data <= counter; -- software reads shared variable
    end if; end if;
  end if;
end process read_shared_variables;
Hardware Process Semantics

hw void write_ab() {
    a = a + 1;
    b = b + a; // sees newest value of a
}

hw void write_c() {
    c = c + b; // sees b delayed by 1 cycle
}

hw void buggy() {
    c = 5;    // ERR: only one process may write
    while (f < 3)    // ERR: loops prohibited
        f = f + 1;
}
shared uint:32 counter;

hw void count() {     // Hardware process
    counter = counter + 1; // Hardware always writes
}

out void reset() {    // Software function
    counter = 0;       // Overrides hardware when executed
}
The I²C Bus

Philips invented the Inter-IC bus c. 1980 as a very cheap way to communicate slowly among chips.

E.g., good for setting control registers.

100, 400, and 3400 kHz bitrates.

SCL: Clock, generated by a single master.

SDA: Data, controlled by either master or slaves.
I2C Bus Transaction

Idle  Start  “0”  “1”  Ack  Stop

SCL  

SDA  

Write data

```
S | slave address | W | A | data | A | data | A | P
< n data bytes >
```

Read data

```
S | slave address | R | A | data | A | data | A | P
< n data bytes >  last data byte
```

Master

transmitter

Slave

receiver

S = Start condition  
A = Acknowledge  
P = Stop condition  
R/W = read / write not  
A̅ = Not Acknowledge
An I²C Bus Controller (Software)

shared out bool SCL;  // I2C clock
shared out bool SDA;   // I2C data out
shared out bool SDA_oe;  // Output enable for data
shared bool SDA_data;  // I2C data in

void send(uint:8 byte) {
    SDA_oe = 0; delay();
    for (int i = 7 ; i >= 0 ; i = i - 1) {
        SDA = (byte & 0x80) >> 7; delay();
        SCL = 1; byte = byte << 1; delay();
        SCL = 0; delay();
    }
    SDA_oe = 1; delay();
    SCL = 1; delay();
    bool acknowledge_received = SDA_data;
    if (!acknowledge_received)
        xio.print("Acknowledge not received\r\n");
    delay();
    SCL = 0; delay();
}
An I²C Bus Controller in Hardware

- One hardware process: state machine plus counters
- One software function per operation (read, write, etc.)
- Shared variables communicate status, data
- Four-phase handshake for synchronization:
  1. Software sends command (e.g., read, write)
  2. Hardware signals “not ready”
  3. Software sends “idle” command
  4. Hardware signals “ready”
An I²C Bus Controller (Hardware)

stared uint:8 sreg;  // Send/receive shift register
shared uint:5 state;  // Controller state
shared bool ready;    // true => controller idling
shared uint:3 command; // Command for the controller
shared const uint:3 IDLE = 0;  // Commands
shared const uint:3 SEND = 2;

hw void controller() {
  const uint:5 IDLE = 0; const uint:5 SEND1 = 5;
  const uint:5 SEND2 = 6; const uint:5 IDLE0 = 24;
  uint:3 bit_counter;

  if (reset) state = IDLE;
  if (i2c_clock) {
    ready = 0;
    switch (state) {
    case IDLE:
      ready = 1;
      switch (command) {
      case START: state = START1; break;
      case SEND: state = SEND1; break;
      case RECEIVE: state = RECV1; break;
      case STOP: state = STOP1; break;
      default: state = IDLE; break;
      }
    break;
  }
case SEND1:
    SDA_oe = 0; bit_counter = 0; state = SEND2;
    break;
case SEND2:
    SDA = sreg[7]; state = SEND3; break;
case SEND3:
    SCL = 1; sreg = sreg << 1;
    bit_counter = bit_counter - 1; state = SEND4;
    break;
case SEND4:
    SCL = 0; if (bit_counter == 0) state = SEND5;
    else state = SEND2; break;
    // Receive Acknowledge
    case SEND5:
    SDA_oe = 1; state = SEND6; break;
case SEND6:
    SCL = 1; state = SEND7; break;
case SEND7:
    acknowledge_received = SDA_in; state = SEND8;
case SEND8:
    SCL = 0; state = IDLE0; break;
case IDLE0:
    if (command == IDLE_BIT) state = IDLE;
    else state = IDLE0; break;
}
}

void send(uint:8 byte) {
    sreg = byte;
    command = SEND_BIT; while (ready) ;
    command = IDLE_BIT; while (!ready) ;
}
## Results

<table>
<thead>
<tr>
<th>Example</th>
<th>SHIM</th>
<th>C</th>
<th>VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>I²C Software (by hand)</td>
<td>259</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>I²C Software</td>
<td>171</td>
<td>175</td>
<td>136</td>
</tr>
<tr>
<td>I²C Bit-level</td>
<td>283</td>
<td>173</td>
<td>337</td>
</tr>
<tr>
<td>I²C Byte-level Receive</td>
<td>299</td>
<td>163</td>
<td>358</td>
</tr>
<tr>
<td>I²C Byte-level Send/Receive</td>
<td>323</td>
<td>116</td>
<td>344</td>
</tr>
<tr>
<td>Timer</td>
<td>15</td>
<td>20</td>
<td>98</td>
</tr>
</tbody>
</table>
Under the hood

SHIM compiler: 3000 lines of OCAML
500 lines: interface synthesis for Xilinx/Microblaze/OPB
Most translation syntax-directed (no optimization)
Enumerate shared variables, assign addresses
Modify software R/Ws of shared variables
Add reads and write ports to hardware processes
What’s Next?

Problem: language is fundamentally nondeterministic

Timer illustrates problem: software runs at rate independent of hardware

Standard solution: simulate hardware in concert with cycle-accurate ISS

I used handshaking to make \( I^2C \) example work reliably

What should the semantics be? I want correct-by-construction