ASIC Verification

Events and Interprocess Communication

Fall 2011
Meeta Yadav
Topics

• Events
• Interprocess Communication
  ◆ Mailboxes
  ◆ Semaphores
• Building a testbench with Threads and Interprocess Communication
Event

• Creating a random transaction

```
[Generator]
↓
Create a transaction
↓
Transaction
↓
Send a tr_ready signal when the transaction is ready
```

- >
Event

- Transaction

```plaintext
class Transaction;
    rand reg[reg_wd-1:0] src1;
    rand reg[reg_wd-1:0] src2;
    rand reg[2:0] alu_operation; int count;
    constraint Limit { src1 inside{[0:65534]};
        src2 inside{[0:65534]};
        alu_operation inside{[1:5]};
    }
endclass
```
Event

- Create a transaction

```cpp
class Generator;
    Transaction tr;
    event tr_ready;
    function new (event tr_ready);
        this.tr_ready=tr_ready;
    endfunction

    task run();
        begin
            tr=new();
            assert(tr.randomize());
        ->tr_ready;
        end
    endtask
endclass
```

Pass event from TB

Create transaction

Tell the TB we are done
Event

• Main Testbench

```verilog
program automatic test;
    event tr_done;
    Generator gen;

    initial begin
        gen=new(tr_done);
        gen.run();
        wait(tr_done.triggered());
    end
endprogram
```

- Create a generator object
- Run Transactor
- Wait for finish
Event

- Creating multiple random transactions

```
Generator

Create multiple transactions

Transaction_1  ->  tr_ready_1

Transaction_2  ->  tr_ready_2

...  ->  ...

Transaction_n  ->  tr_ready_n
```

Send a `tr_ready` signal when the transaction is ready
Event

• Generator can now handle multiple transactions

```plaintext
class Generator;
    Transaction tr;
    event tr_ready;
    function new (event tr_ready);
        this.tr_ready=tr_ready;
    endfunction

task run();
    fork
        begin
            tr=new();
            assert(tr.randomize());
            ->tr_ready;
        end
    join_none
endtask
endclass
```

- Pass event from TB
- Create transactions
- Tell the TB we are done
Event

• Creating random transactions

- Generate multiple instructions
- Create multiple transactions
  - Transaction_1
  - Transaction_2
  - …
  - Transaction_n

- Send a tr_ready signal when the transaction is ready
  - tr_ready_1
  - tr_ready_2
  - …
  - tr_ready_n
Event

- Waiting for multiple events

```
event tr_ready[N_GENERATORS];
Generator gen[N_GENERATORS];
initial begin
  foreach (gen[i]) begin
    gen[i]=new(tr_ready[i]);
    gen[i].run();
  end
  foreach(gen[i])
    fork
      automatic int k=i;
      wait(tr_ready[k].triggered());
    join_none
    wait fork;
  end
```

*Waiting for multiple threads with wait fork*
Event

• Waiting for multiple events

Start parallel threads to create transactions and trigger tr_ready

Wait on these parallel threads
Interprocess Communication: Mailboxes

How do you pass information between two threads?
Interprocess Communication: Mailboxes

• **What is a mailbox?**
  - Mailbox is like a FIFO
  - Source puts value in the mailbox
  - Sink removes values from the mailbox

![Diagram of a mailbox with source, sink, and FIFO structure.]
Interprocess Communication: Mailboxes

- **How do you create a mailbox?**
  - Mailbox is an object
  - Need to call `new` to instantiate it
  - Mailbox can be unbounded (unlimited size) or bounded (limited size)
  - New takes mailbox size as an argument
    - If unspecified means unbounded

```
mailbox GenDrv;
GenDrv=new(4);
```

*Mailbox of size 4*
Interprocess Communication: Mailboxes

- **What can you put in the mailbox?**
  - Single value data for example an integer or logic (can be any size)
  - Can also put handles into the mailbox
    - Cannot put objects into the mailbox
  - Can put a mix of data
    - DON’T DO IT!!!!
Interprocess Communication: Mailboxes

- How do you put data in the mailbox?
  - Use the `put` task
    - If the mailbox is full `put` blocks
  - Put data into the mailbox without blocking
    - `try_put()`

```
mailbox GenDrv;
GenDrv=new(4);
GenDrv.put(t);
```

Use `put` task to put data in the mailbox
Interprocess Communication: Mailboxes

• How do you get data from the mailbox?
  ◆ Use the get task
    ➢ get removes the data from the mailbox
    ➢ If the mailbox is full get blocks
  ◆ Use the peek task
    ➢ peek gets the data but does not remove it from the mailbox
    ➢ If the mailbox is full peek blocks
  ◆ Use the try_peer() task and the try_get() task to not block while retrieving data from the mailbox

```
mailbox GenDrv;
GenDrv=new(4);
GenDrv.put(t);
GenDrv.get(t);
```

Use put task to put data in the mailbox
Use get task to get data from the mailbox
Interprocess Communication: Mailboxes Summary

• Mailboxes
  ◆ A mailbox is a communication mechanism that allows messages to be exchanged between processes. Data can be sent to a mailbox by one process and retrieved by another
  ◆ From a hardware point of view a mailbox is nothing but a FIFO with a source and a sink
    ▶ The source puts the data into the mailbox and the sink gets values from the mailbox
  ◆ Mailboxes can have a maximum size or can be unlimited
    ▶ Mailboxes are created either having a bounded or unbounded queue size
    ▶ A bounded mailbox becomes full when it contains the bounded number of messages
    ▶ A process that attempts to place a message into a full mailbox shall be suspended until enough room becomes available in the mailbox queue
    ▶ If a sink tries to retrieve data from a mailbox that is empty it blocks it until data is put in the mailbox
Interprocess Communication: Mailboxes Summary

• Mailboxes
  ✷ Mailbox is an inbuilt class that provides the following methods:
    ▶ Create a mailbox: new()
    ▶ Place a message in a mailbox: put()
    ▶ Try to place a message in a mailbox without blocking: try_put()
    ▶ Retrieve a message from a mailbox: get() or peek()
    ▶ Try to retrieve a message from a mailbox without blocking: try_get()
      or try_peek()
    ▶ Retrieve the number of messages in the mailbox: num()
Interprocess Communication: Mailboxes

- Mailbox Example
  - Randomize objects and put handles to the object in the mailbox

```hsql
task generator(int n, mailbox mbx);
 Transaction t;
 t=new();
 repeat (n) begin
   assert(t.randomize());
   mbx.put(t);
 end
endtask
```

*Example: Randomizing objects and sending them to the driver*
Event

- Creating random transactions and sending them through mailboxes

All handles in the mailbox point to the same object
Interprocess Communication: Mailboxes

• Mailbox Example
  ◆ Randomize objects and put handles to the object in the mailbox

```verbatim
task generator(int n, mailbox mbx);
  Transaction t;
  repeat (n) begin
    t=new();
    assert(t.randomize());
    mbx.put(t);
  end
endtask
```

Example: Randomizing objects and sending them to the driver
Event

- Creating random transactions and sending them through mailboxes

All handles in the mailbox point to different object
Interprocess Communication: Mailboxes

- Mailbox in a testbench
  - How do you pass information between two threads?
    - The generator creates many transactions and passes them to a driver
    - The generator and driver must operate asynchronously

```plaintext
class Generator
  Transaction tr;
  mailbox mbx;

  function new(mailbox mbx);
    this.mbx=mbx;
  endfunction

  task run (int count);
    repeat (count) begin
      tr=new();
      assert(tr.randomize);
      mbx.put(tr);
    end
  endtask
endclass
```

Example: Exchanging objects using a mailbox: the Generator class
Interprocess Communication: Mailboxes

• Mailbox in a testbench

```class` Driver
  Transaction tr;
  mailbox mbx;

  function new(mailbox mbx);
    this.mbx=mbx;
  endfunction

  task run(int count);
    repeat (count) begin
      mbx.get(tr);
      @(posedge busif.cb.ack)
      ...
    end
  endtask
endclass
```

*Example: Example of using a mailbox in a testbench*
Interprocess Communication: Mailboxes

• Mailbox in a testbench

```
program automatic mailbox_example (...)  
  `include “transaction.sv”  
  `include “generator.sv”  
  `include “driver.sv”

mailbox mbx;  
Generator gen;  
Driver drv;  
initial begin  
  count = $urandom(50)  
  mbx=new();  
  gen=new(mbx);  
  drv=new(mbx);  
  fork  
    gen.run(count);  
    drv.run(count);  
  join  
end
endprogram
```

Example: Example of using a mailbox in a testbench, continued
Interprocess Communication: Mailboxes

- Unsynchronized threads communicating with the testbench

```plaintext
program automatic unsynchronized;
    class Producer;
    task run;
        for(int i=1; i<4; i++) begin
            $display("Producer: before put(%0d)",i);
            mbx.put(i);
        end
    endtask
endclass

class Consumer;
    task run;
        int i;
        repeat(3) begin
            mbx.get(i);
            $display("Consumer: after get(%0d)",i);
        end
    endtask
endclass

initial begin
    mbx=new;
p=new;c=new;
    fork
        p.run;
c.run;
    join
end
endprogram
```

Example: Producer-consumer without synchronization
### Interprocess Communication: Mailboxes

- Unsynchronized threads communicating with the testbench

```verilog
program automatic unsynchronized;
    class Producer;
        task run;
            for(int i=1; i<4; i++) begin
                $display("Producer: before put(\%0d)",i);
                mbx.put(i);
            end
    endtask
endclass

class Consumer;
    task run;
        int i;
        repeat(3) begin
            mbx.get(i);
            $display("Consumer: after get(\%0d)",i);
        end
    endtask
endclass

mailbox mbx;
Producer p;
Consumer c;
initial begin
    mbx=new;p=new;c=new;
    fork
        p.run;
        c.run;
    join
end
endprogram
```

There is no synchronization so the producer puts all three integers into the mailbox before the consumer can get the first one. This is because the thread continues running until there is a blocking statement and the Producer has none.

Example: Producer-consumer without synchronization
Interprocess Communication: Mailboxes

- Bounded Mailboxes

```verilog
program automatic bounded;
    mailbox mbx;
    initial begin
        mbx = new(1);
        fork
            for (int i=1; i<4; i++) begin
                $display("@%0d: Producer: putting %0d", $time, i);
                mbx.put(i);
                $display("@%0d: Producer: put(%0d) done %0d", $time, i);
            end
        repeat(3) begin
            int j;
            #1ns mbx.get(j);
            $display("@%0d: Consumer: got %0d", $time, j);
        end
        join_any
    end
endprogram
```

**Example: Bounded mailbox**
Interprocess Communication: Mailboxes

• Bounded Mailboxes

```
program automatic bounded;
mailbox mbx;
initial begin
  mbx = new(1);
  fork
    for (int i=1; i<4; i++) begin
      $display("@%0d: Producer: putting %0d", $time, i);
      mbx.put(i);
      $display("@%0d: Producer: put(%0d) done %0d", $time, i);
    end
  repeat(3) begin
    int j;
    #1ns mbx.get(j);
    $display("@%0d: Consumer: got %0d", $time, j);
  end
  join_any
end
```

**Example: Bounded mailbox**

- **Producer**: putting 1
  - @0 Producer: putting 1
  - @0 Producer: put(1) done
- **Producer**: putting 2
  - @0 Producer: putting 2
  - @1 Producer: put(2) done
- **Producer**: putting 3
  - @1 Producer: putting 3
  - @2 Producer: put(3) done
- **Consumer**: got(1)
  - @1 Consumer: got(1)
- **Consumer**: got(2)
  - @2 Consumer: got(2)
- **Consumer**: got(3)
  - @3 Consumer: got(3)

Producer runs ahead of consumer!!!
Interprocess Communication: Mailboxes

• Synchronized threads using a mailbox and events
  ◆ An additional handshake is required if the producer and the consumer are to be run in lock-step

```
program automatic mbx_evt;
  event handshake;
  class Producer;
    task run;
      for(int i=1; i<4; i++) begin
        $display(“Producer: before put(%0d)”,i);
        mbx.put(i);
        @handshake;
        $display(“Producer: after put(%0d)”,i);
      end
    endtask
  endclass
  class Consumer;
    task run;
      int i;
      repeat(3) begin
        mbx.get(i);
        $display(“Consumer: after get(%0d)”,i);
        -> handshake;
      end
    endtask
  endclass
...
endprogram
```

The two threads should use a handshake so that the Producer does not get ahead of the Consumer. This is done by blocking the Producer on the `handshake` event (the consumer already blocks on the mailbox)

Producer blocks on the `handshake` event to ensure that the producer stops after sending the transaction

Consumer triggers the `handshake` event to allow the Producer to advance

Example: Producer-consumer synchronized with an event
Interprocess Communication: Mailboxes

• Synchronized threads using a mailbox and events

```
program automatic mbx_evt;
    event handshake;
    class Producer;
        task run;
            for(int i=1; i<4; i++) begin
                $display(“Producer: before put(%0d)”,i);
                mbx.put(i);
                @handshake;
                $display(“Producer: after put(%0d)”,i);
            end
        endtask
    endclass
    class Consumer;
        task run;
            int i;
            repeat(3) begin
                mbx.get(i);
                $display(“Consumer: after get(%0d)”,i);
                -> handshake;
            end
        endtask
    endclass
endprogram
```

Example: Producer-consumer synchronized with an event
Interprocess Communication: Mailboxes

• Synchronized threads using two mailboxes

```verbatim
program automatic mbx_mbx2;
    mailbox mbx, rtn;
    class Producer;
        task run;
            for (int i=1; i<4; i++) begin
                $display(“Producer: before put(%0d)”,i);
                mbx.put(i);
                rtn.get(i);
                $display(“Producer: after put(%0d)”,i);
            end
        endtask
    endclass
    class Consumer;
        task run;
            int i;
            repeat(3) begin
                mbx.get(i);
                $display(“Consumer: after get(%0d)”,i);
                rtn.put(i)
            end
        endtask
    endclass
endprogram
```

Example: Producer-consumer synchronized with an event
Building a Testbench with Threads and IPC

![Diagram of Testbench with Threads and IPC]

Testbench
Building a Testbench with Threads and IPC

- Basic Transactor

```verilog
class Agent;
  mailbox gen2agt, agt2drv;  // Create mailboxes to send transactions from generator to agent, agent to driver
  Transaction tr;
  function new(mailbox gen2agt, agt2drv);
    this.gen2agt = gen2agt;
    this.agt2drv = agt2drv;
  endfunction
  function build;
    // Empty for now
  endfunction
  task run;
    forever begin
      gen2agt.get(tr);  // Get transaction from upstream block
      //Do some processing
      agt2drv.put(tr);  // Send transaction to downstream block
    end
  endtask
  task wrapup;
    // Empty for now
  endtask
endclass
```

Example: Basic Transactor for Agent that sits between Generator and Driver
Building a Testbench with Threads and IPC

- Environment Class
  - The Generator, Agent, Monitor, Checker and Scoreboard classes are instantiated in the Environment class

```plaintext
class Environment;
  Generator gen;
  Agent agt;
  Driver drv;
  Monitor mon;
  Checker chk;
  Scoreboard scb;
  Config cfg;
  mailbox gen2agt, agt2drv, mon2chk;
  extern function new;
  extern function void gen_cfg;
  extern function void build;
  extern task run;
  extern task wrapup;
endclass

function Environment::new;
  gen2agt = new;
  agt2drv = new;
  mon2chk = new;
  gen = new(gen2agt);
  agt = new(gen2agt, agt2drv);
  drv = new(agt2drv);
  mon = new(mon2chk);
  chk = new(mon2chk);
  scb = new;
  cfg = new;
endfunction
```

 Instantiate Generator, Agent, Monitor, Checker and Scoreboard classes

Create mailboxes to send transactions from generator to agent, agent to driver and monitor to checker

Run Generator, agent, driver, monitor and checker in parallel

```plaintext
function void Environment::gen_cfg;
  assert(cfg.randomize);
endfunction
function void Environment::build;
  gen.build;
  agt.build;
  drv.build;
  mon.build;
  chk.build;
  scb.build;
endfunction

task Environment::run;
  fork
    gen.run(run_for_n_trans);
    agt.run;
    drv.run;
    mon.run;
    chk.run;
    scb.run(run_for_n_trans);
  join
endtask

task Environment::wrapup;
  fork
    gen.wrapup;
    agt.wrapup;
    drv.wrapup;
    mon.wrapup;
    chk.wrapup;
    scb.wrapup;
  join
endtask
```

Example: Environment Class

Initialize mailboxes

Initialize transactors

Example: Environment Class, continued
Building a Testbench with Threads and IPC

• Test Program
  ◆ The main test goes in the top level program

```plaintext
program automatic test;
    Environment env;
    initial begin
        env = new;
        env.gen_cfg;
        env.build;
        env.run;
        env.wrapup;
    end
endprogram
```

*Example: Basic Test Program*
Thank You