ASIC Verification

SystemVerilog Assertions 2

Fall 2011
Meeta Yadav
Topics

- Properties
- Sequences and Delays
- Regular Expressions
- Implication Operators
Property

- Properties define the behavior of the design
- SVAs are represented as properties
- A property is a rule that will be asserted passively to test the design
- Property can be:
  - A simple Boolean test regarding conditions that always hold true in the design
  - Property can also be a sampled sequence of signals that should follow a legal and prescribed protocol
    - A complex sequence can be partitioned into simple sequence blocks

```verilog
property pReqAck;
 @(posedge clk) request |-> ## [1:3] acknowledge;
endproperty: pReqAck
apReqAck : assert property (pReqAck)
```

Example of asserting a property

```verilog
sequence s1;
   @(posedge clk) a ##1 b ##[0:3] c;
endsequence
```

Example of a sequence
Sequences and the ## Delay

◆ A sequence is a series of true/false expressions spread over one or more clock cycles

◆ ## represent a “cycle delay”
  ▪ Specified the number of clock cycles to wait until the next expression in the sequence if evaluated
    • The first expression is evaluated immediately
    • Subsequent expressions are evaluated specified clock cycles later

\[
\text{req } ##1 \text{ gnt } ##1 \text{ !req}
\]

Example of a sequence
Example of ## Delay

 req should be true on the current clock tick
 req #1 gnt #1 !req

gnt should be true on the subsequent clock tick

req shall be false on the next clock tick after that

Example of a sequence

req high not detected
req high detected
req low detected

s0
s1
s2

clk

req

gnt
Example of ## Delay

sequence s1;
    @(posedge clk) a ##1 b ##1 c;
endsequence

sequence s2;
    @(negedge clk) g ##1 h ##1 i;
endsequence

Example of sequences
Multiple Clock Cycle Delays

◆ ##n specifies a fixed number of clock cycles
  ▪ n must be a non negative constant expression

```
req ##3 gnt
```

*Example of ##n

After evaluation request, skip 2 clock cycles and evaluate grant on the third clock cycle

◆ ##[min_count:max_count] specifies a range of clock cycles
  ▪ min_count and max_count must be non negative constants

```
req ##[1:3] gnt
```

*Example of ##[min_count:max_count]

After evaluating request, grant must be true between 1 and 3 clocks later
Example of \texttt{#\#n} Delay

```verilog
sequence sl;
    @(posedge clk) a #1 b #[0:3] c;
endsequence
```

The sequence requires that \texttt{c} occurs 0, 1, 2, or 3 cycles after \texttt{b}.

Possible Solutions for \texttt{c}
Infinite Cycle Delays

◆ ($) specifies infinite number of clock cycles
  ▪ n must be a non negative constant expression

```
req ##[1:$] gnt
```

Example of $:

Request must be true and then grant must become true sometime between now and end of time

◆ In simulation end of time is end of simulation
  ◆ Report an assertion as a failure or
  ◆ Report an assertion as an uncompleted assertion
Example of $\phi$ Delay

sequence s1;
  @(posedge clk) a ##1 b ##[0:$] c;
endsequence

The sequence requires that $c$ occurs 0, 1, 2… or infinite cycles after $b$.
Quiz

```verilog
sequence sl;
  a ##1 b ##1 c;
endsequence

sequence rule;
  @(posedge clk) d ##1 e ##1 sl ##1 f;
endsequence
```

**Example of a sequence**
Repeated Regular Expressions

- A sequence of events can be repeated using *consecutive repetition* \([*n]\) operator
  - \(n\) must be non negative

\[
\begin{align*}
a & \textcircled{#1} b \textcircled{#1} b \textcircled{#1} b \textcircled{#1} c \\
\text{Can be written as } & b[*3]
\end{align*}
\]
Repeated Regular Expressions

◆ A range of steps can be repeated using a count in the form of
  
  \([*]^{\text{min\_count}:\text{max\_count}}\)

  ▶ must be non negative constants

\[
\begin{align*}
a & \quad b \quad [*2:4] \quad c \\
\end{align*}
\]

Equivalent to

\[
\begin{align*}
a & \quad b \quad b \quad c \\
\end{align*}
\]

\[
\begin{align*}
a & \quad b \quad b \quad b \quad c \\
\end{align*}
\]

\[
\begin{align*}
a & \quad b \quad b \quad b \quad b \quad c \\
\end{align*}
\]

\[
\begin{align*}
a & \quad b \quad b \quad b \quad b \quad b \quad c \\
\end{align*}
\]
Sequences

- Sequence Operations: Consecutive Repition

Example of a sequence

```
a  ##1 b  [*3]  ##1  c
```

Waveform for the sequence
Sampled Value Functions

- $\text{sampled}$: Returns the sampled value of the expression with respect to the last occurrence of the clocking event.
- $\text{rose}$: Returns true if the least significant bit of the expression changed to 1. Otherwise it returns false.
- $\text{fell}$: Returns true if the least significant bit of the expression changed to 0. Otherwise it returns false.
- $\text{stable}$: Returns true if the value of the expression did not change. Otherwise it returns false.

![Diagram showing simulation ticks and clock ticks with sequences for req, ack, $\text{rose}(\text{req})$, and $\text{fell}(\text{ack})$.]
Sampled Value Functions

- **Sampled value functions**
  - $\textit{past}$: returns the sampled value of the expression that was present \textit{number_of_ticks} prior to the time of evaluation of $\textit{past}$.  

```verilog
always @(posedge clk)
    reg1 <= a & $past(b);
```

**Example of $\textit{past}$**

![Sampled Value Function Diagram](image-url)
Conditioning Sequences Using Implication Operators

- Evaluation of a sequence can be preconditioned with an implication operator
  - $\rightarrow$ Overlapped
    - If the condition is true, sequence evaluation starts immediately
    - If the condition is false, the sequence acts as if it succeeded
  - $\Rightarrow$ Non-overlapped:
    - If the condition is true, sequence evaluation starts in the next clock cycle
    - If the condition is false, the sequence acts as if it succeeded

```verilog
property p_handshake;
@ (posedge clk)
request $\Rightarrow$ acknowledge ##1 data_enable ##1 done;
endproperty

aphandshake: assert property (p_handshake);
```

Property `p_handshake` states that if the antecedent sequence `request` is TRUE then the consequent must also be TRUE otherwise the property fails.
Example of Implication Operator

```verilog
property p_handshake;
  @(posedge clk)
  request |=> acknowledge ##1 data_enable ##1 done;
endproperty
aphandshake: assert property (p_handshake);
```

Two active threads T1 and T2 for the handshake property
Example of Implication Operator

- Example

```
property data_end;
   @(posedge mclk)
      data_phase |-> ((irdy==0) && ($fell(trdy) || $fell(stop)))
endproperty
```

Example of conditional sequence matching
Implication

- Implication

```verbatim
'define data_end_exp (data_phase && ((irdy==0)&&(fell(trdy)||fell(stop))))

property data_end_rule1;

@ (posedge mclk) 'data_end_exp |-> ##[1:2] rose(frame) #1 rose(irdy);

endproperty
```

Example of conditional sequences
Sequences: AND Operation

- The binary operator \textit{and} is used when both operands are expected to match but the end times of the operand sequences can be different.
  - The operand sequences start at the same time. When one of the operand sequence matches, it waits for the other to match. The end time of the composite sequence is the end time of the operand sequence that completes last.

  - Matches is te1 and te2 match
  - The end time is the end time of either te1 or te2, whichever matches last

\[\text{te1 and te2}\]

\textit{ANDing (and) two boolean expressions}
Sequences: Example of AND Operation

- **AND operation**
  - The two operands of “and” are sequences. The requirement for the match of the and operation is that both the operands must match.

\[(te_1 \ #\ #\ 2 \ te_2) \text{ and } (te_3 \ #\ #\ 2 \ te_4 \ #\ #\ 2 \ te_5)\]

*Example of and operation*

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+

ANDing (and) two sequences
Sequences: AND Example

\[ (te_1 \#\#[1:5] \, te_2) \] and \[ (te_3 \#\#2 \, te_4 \#\#2 \, te_5) \]

\textbf{ANDing (and) two sequences, including a time range}
Sequences: Intersect

- Intersection (AND with length restriction)
  - The binary operator \texttt{intersect} is used when both operand sequences are expected to match, and the end times of the operand sequences must be the same.
    - Both the operands must match.
    - The lengths of the two matches of the operand sequences must be the same.

\texttt{tel intersect te2}
Sequences: Intersect Example

- Intersect operation

\[(\text{te1} \;++[1:5] \;\text{te2}) \;\text{intersect} \;\left(\text{te3} \;++2 \;\text{te4} \;++2 \;\text{te5}\right)\]

*Example of intersect operation*

Intersecting two sequences
**Sequences: OR**

- **Or operation**
  - The operator `or` is used when at least one of the two operand sequences is expected to match.
  - If the operands `te1` and `te2` are expressions, then `te1 or te2` matches at any clock tick on which at least one of `te1` and `te2` evaluates to true.

![Diagram showing ORing two boolean expressions](image)

*ORing two boolean expressions*
Sequences: OR Example

- OR operation

Example of OR operation

ORing two sequences
Sequences: OR Example

- OR operation

\[
(te_1 \text{##}[1:5] \ te_2) \ or \ (te_3 \text{##}2 \ te_4 \text{##}2 \ te_5)
\]

**Example of or operation**

**ORing two sequences, including a time range**
Sequences: **throughout** Example

- **Conditions over sequences**

```verilog
sequence burst_rule1;
    @(posedge mclk)
        $fell(burst_mode) ##0
        (!burst_mode) throughout (#2 ((trdy==0)&&(irdy==0)) [*7]);
endsequence
```

**Example of throughout**

![Diagram showing sequence conditions and matches](image)

*Match with throughout restriction fails*
Sequences: \textbf{throughout Example}

- Conditions over sequences

```verbatim
sequence burst_rule1;
  @(posedge mclk)
    $fell(burst_mode) ##0 (!burst_mode) throughout (#2 ((trdy==0)&&(irdy==0)) [*7]);
endsequence
```

\textbf{Example of throughout}

\begin{center}
\begin{tabular}{ccccccccccccccc}
  mclk & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
  burst_mode & & & & & & & & & & & & & & \\
  irdy & & & & & & & & & & & & & & \\
  trdy & & & & & & & & & & & & & & \\
  (trdy==0) && (irdy==0) & & & & & & & & & & & & \\
  burst_rule1 & & & & & & & & & & & & & & \\
\end{tabular}
\end{center}

\textbf{Match with throughout restriction passes}
Thank You