Architectures for Computer Vision
from Algorithm to Chip with Verilog

Hong Jeong

©2014 John Wiley & Sons Singapore Pte Ltd. Published 2014 by John Wiley & Sons Singapore Pte Ltd.

March 13, 2015
Part 1 Verilog HDL
Chapter 2 Verilog HDL, Communication, and Control
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Figure: The Verilog system: TB-UUT modules.
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Hello world in Verilog

module main;
    initial
    begin
        $display("Hello, world!
");
        $finish;
    end
endmodule

Hello world in C

#include<stdio.h>
main();
{
    printf("Hello, world!
");
}
Listing 1: A 4-bit adder: adder.v

```verilog

'timescale 1ns/1ps //unit time/precision
module adder( //ports
    input [3:0] a, b, //input ports
    output [3:0] c //output ports
);

assign c= a+ b; //continuous assignment
endmodule
```

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.) Architectures for Computer Vision
Listing 2: A test bench: `tb.v`

```verilog
`timescale 1ns/1ps
module tb; //no ports
   //declaration
   reg [3:0] a, b; //reg type for storage
   wire [3:0] c; //wire for connection
   //instantiation
   adder UUT (.a(a),.b(b),.c(c)); //run UUT
   //test vector generation
   initial begin //run once for simulation
      //initialize Inputs
      a = 0; //execute sequentially
      b = 0;
      #100; //wait 100 ns
      //add stimulus here
      repeat (1000) begin //repeat 1000 times
         a = a + 1; //execute sequentially
         b = b + 2;
         #100; //wait 100ns
      end
   end
endmodule
```

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.)
Architectures for Computer Vision
Figure: The simulator output: timing diagram.
Contents

The Verilog System
Hello, World!

Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Figure: A hierarchy of modules.
Figure: Connecting two modules by ports.
Listing 3: Module constructs

module module_name (port-name, port-name,...,port-name)
    //port declarations
    input declarations    //port directions
    output declarations
    inout declarations

    //type declarations
    net declarations      //data and variable declarations
    variable declarations
    parameter declarations //parameter declarations

    //functions and tasks
    function declarations //function definition
    task declarations    //task definition

    //execute once for TB
    initial begin         //one-time execution statements
        instantiations   //instantiation of other modules
    end

    //procedural statements
always begin //statements for a design
  procedural statements
end
endmodule
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 4: The test bench constructs

module testbench_name
  instantiation //instantiation of UUT

  initial //one time execution
    begin
      procedural-statement //test vector generation
        ...
      procedural-statement //checking and
      end
      procedural-statement //report
  end
endmodule
(a) The connection of UUT-TB  
(b) The TB structure

Figure: The UUT and the TB.
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 5: Arrays

[MSB_1:LSB_1]...[MSB_n:LSB_n] variable_identifier
[MSB_1:LSB_1]...[MSB_m:LSB_m]
Example (Arrays)

Examples of arrays are as follows:

```vhdl
reg a[7:0]; // 8 1-bit scalar register
reg [7:0] b; // 1 8-bit vector register
reg c[7:0][0:255]; // 8 x 256 array of 1-bit
reg [0:7] d [0:255]; // 256 8-bit vector indexed from 0 to 7

// The followings are allowed only in SystemVerilog.
reg [1:3][7:0] e; // 24-bit 3-field vector
reg [1:3][7:0] f[0:255] // 256 24-bit 3-field vectors
reg [1:3][7:0] g[1:2][0:255] // 512 24-bit 8-field vectors
```
Example (Strengths and delays)
Some typical examples are as follows.

```plaintext
triereg a; //charge strength medium
triereg (small) #(0,0,100) b; //charge strength and delay
triereg (large) unsigned [0:7] c; //charge with range
and #(10) and1 (out,input1,input2); //delay
and #(10,20) and2 (out,input1,input2); //delay
bufif0 #(1,2,3) buff0 (i01,i02,dir); //delay
bufif0 #(1:2:3,4:5:6,7:8:9) buff1 (io1, io2, dir); //delay
```
Example (Expressions)

Some examples are as follows.

```
&4'b1001=0  // reduction
false: 4'b0000!, true: 4'b0010!  // logic value
{2'b10,2'b01} = 4'b1001  // concatenation
4'b0100 & 4'b01xz = 4'b0100  // bit-wise logic
~2'b10 = 2'b01  // bit-wise complement
16'b0,8'bz01 = 8'bzzzzzzz01  // bit-wise
true: 2'b10 < 4'b010  // logic statement
2'h06 == 4'b0110  // logic statement
X ? Y:Z  // if X is true then Y, else Z
```
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Example (Continuous assignments)
The two expressions are effectively the same.

Continuous declaration

```plaintext
wire (strong1, pull0) b = a;
```

Declaration, assignment

```plaintext
wire b;
assign (strong1, pull0) b = a;
```
Example (Delays)

The continuous assignment with delay.

wire #100 a;
assign wire c = (#20) a + b;
Example (Strengths)

The strengths for a continuous assignment.

assign (strong1, pull0) b = a;  //the same as below
assign (pull0, strong1) b = a;
assign (pull0, pull1) b = a;  //wrong
Example (Blocking and nonblocking assignments)
Swapping values.

Blocking statements
always @(posedge clock) begin
  c = a; //temporary variable c
  a = b;
  b = c;
end //always

Nonblocking statements
always @(posedge clock) begin
  b <= a; //RHS for the 1st step
  a <= b; //LHS for the 2nd step
end //always
Example (assign-deassign)

The procedural continuous assignment.

```verilog
always @(posedge clock)
    Count = Count + 10; //Count generation
always @(reset or set)
    if (reset) //asynchronous reset
        assign Count = 0; //prevents counting, until reset goes low
    else if (set) //asynchronous set
        assign Count = 1; //prevents counting, until set goes low
    else
        deassign Count; //resume counting on next posedge clock

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.)
Architectures for Computer Vision
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments

Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 6: Instantiation

component-name instance_identifier (expr, expr, ..., expr);
Example (Structure of module)

An inhibition gate can be built as follows.

```verilog
module Inhibitor (in, invin, out); // BUT-NOT
    input in, invin;                // port declaration
    output out;
    wire notinvin;                 // variable
    not Q1 (.out(notinvin),.in(invin)); // instantiation
    and Q2 (out,in,notinvin);      // instantiation
endmodule
```
Listing 7: Procedural assignments

Blocking assignment: variable-name = expression
Non-blocking assignment: variable-name <= expression
Listing 8: Scopes

begin: block-name
  variable declaration
  parameter declarations
  procedural statements
end
Listing 9: always block

always @(signal-name or signal-name) procedural statements
Listing 10: Conditionals

```plaintext
if (condition) procedural-statement  //if condition
else procedural-statement

case (selection-expression)  //case statement
  choice,...,choice: procedural-statements
  ...
  choice,...,choice: procedural-statement
endcase

for (loop-index=first, loop-index <= last);  //for loop
  loop-index = loop-index+1;
  procedural-statement

repeat (index-expression) procedural-statement  //repeat loop

while (logical-expression) procedural-statement  //while loop

forever procedural-statement  //forever loop

fork procedural-statement  //fork statement
```
join
Example (fork/join)

A simple example.

```verbatim
initial fork
   $write("A");          //print A
   $write("B");          //print B
   begin #5              //wait 5 time units
      $display("C");    //display C
   end
join
```
Example (Timing control)

The timing control example.

```vhdl
#100 b = a; //delay 100 time units
@c b = a; //at the change of c
@ (posedge clock) b = a; //positive edge of clock
always @(a or b, c) d = a+b+c; //event logical
always @(*) c = a+b; //equivalent to @(a or b)
always @* begin
    c = a; d = b; e = c + d;
end
wait (!enable) b = a; //at the change of enable
```
Example (Intra-assignment)
The example is as follows.

\[
\text{always @Swap} \\
\text{fork} \\
\begin{align*}
\text{#10 } a & = b; // \text{at 10,} \\
\text{#10 } b & = a; // a=b \text{ and } b=a
\end{align*}
\text{join}
\]

\[
\text{always @Swap} \\
\text{fork} \\
\begin{align*}
a & = \text{#10 } b; // \text{at 0, tmp1=b, tmp2=a} \\
b & = \text{#10 } a; // \text{at 10, a=tmp1, b=tmp2}
\end{align*}
\text{join}
\]
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements

Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 11: Tasks and functions

task task_name;
    input declarations
    output declarations
    variable declarations
    parameter declarations
    procedural-statements
endtask

function result-type function-name;
    input declarations
    variable declarations
    parameter declarations
    procedural-statements
endfunction
Example (Factorial)
The use of tasks and functions in factorial.

### Task

```verilog
module factorial_task;
  //define the task
  task factorial (level,result);
    input [31:0] level;
    output integer result;
    integer i;
    if (level >= 2)
      factorial (level-1,result);
    result = result * level;
  else
    result = 1;
endtask
```

//test the task
integer result;
integer n;
initial begin
  for (n = 0; n <= 7; n = n+1)
    begin
      factorial(n,result);
      $display("%0d factorial=%0d", n, result);
    end
endmodule //factorial_task
```

### Function

```verilog
module factorial_function;
  //define the function
  function automatic integer factorial;
    input [31:0] level;
    integer i;
    if (level >= 2)
      factorial = factorial (level - 1) * level;
    else
      factorial = 1;
  endfunction
```

//test the function
integer result;
integer n;
initial begin
  for (n = 0; n <= 7; n = n+1)
    begin
      result = factorial(n);
      $display("%0d factorial=%0d", n, result);
    end
endmodule //factorial_function
```
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions

Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 12: Summary of syntax

module arch_vision (Q1,Q2,Q3,Q4);
  //declaration
  input Q1,Q2;  //ports
  output[7:0] Q3;
  inout Q4;
  reg[&:0] Reg1,Mem1[1:254];  //variables
  wire Wire1,Wire2,Wire3,Wire4;
  parameter String = "vision architecture";
  //one time execution for Test bench
  initial  //for simulation only
    begin: BlockName
      Statements
    end
  //continuous assignments
  assign Wire1 = Expression;
  assign wire[3:0] Wire2 = Expression;
  //procedural assignments
  always @(sensitivity-list)
    begin
      procedural-statements;
    end
//module instances, COMP defined in other module
COMP Q (Wire3,Wire4);                          //external module call
Task(.A(Q1),.B(Q4),.C(Q3));                   //procedure call
Q3 = Function(Q1);                              //function call

//procedures definition
task Task:
    input A;                                   //ports
    inout B;
    output C;
    begin                                       //main part
        Statements
    end
endtask

//function definition
function[7:0] Function;
    input A;                                   //ports
    begin
        Function = Expression;                  //use Function <= RHS
    end
endfunction
endmodule
Listing 13: Summary of statements

```plaintext
#delay //delay expressions
wait (Expression)
@(A or B or C) //triggering statements
@(posedge Clk)
Reg = Expression; //assignment statements
Reg <= Expression;
assign Reg = Expression;
deassign Reg;
TaskEnable(...); //event control
disable TaskOrBlock;
->EventName; //event trigger
```
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary

Simulation-Synthesis

Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis

Verilog System Tasks and Functions

Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 14: Display-write tasks

display_task_name (list_of_arguments);
Example (Display task)

Examples of display tasks.

\$write ("value = %b", bval); //binary data
\$write ("value = %0d", dval); //suppress leading zeroes
\$display ("value = %0", dval); //dval is written in octal
\$display ("time = %t", $time); //simulation time
\$strobe ("time = %0d", dval); //at each simulation time
\$monitor ("value = %0d", dval); //displays at dval change
Listing 15: File open-close

multi_channel_descriptor = $open ("file_name");
file_descriptor = $open ("file_name", type);
fclose (multi_channel_descriptor);
fclose (file_descriptor);
Listing 16: File output

file_output_task_name (multi_channel_descriptor,list_of_arguments);
file_output_task_name (file_descriptor, list_of_arguments);
Listing 17: String output

```verilog
$swrite (output_reg, list_of_arguments);
$sformat (output_reg, format_string, list_of_arguments);
```
Listing 18: Character-string-text input

character = $fgetc (file_descriptor);
code = $ungetc (character, file_descriptor);
code = $fgets (reg, file_descriptor);
code = $fscanf (file_descriptor, format, arguments);
code = $sscanf (reg, format, arguments);
code = $fread (dest, file_descriptor, start, count);
$readmemb (file_reg, memory_name, start, finish);
$readmemh (file_reg, memory_name, start, finish);
**Listing 19: File positioning**

```verilog
position = $ftell (file_descriptor);
code = $fseek (file_descriptor, offset, operation);
code = $rewind (file_descriptor);
```
Example (File I/O)

The I/O examples are as follows.

```verilog
integer messages, broadcast,  
    r_color, g_color, b_color;  
initial begin  
    r_color = $fopen("r.dat");  
    if (r_color == 0) $finish;  
    g_color = $fopen("alu.dat");  
    if (g_color == 0) $finish;  
    b_color = $fopen("mem.dat");  
    if (b_color == 0) $finish;  
    messages = r_color|g_color|b_color;  
    //broadcast std output  
    broadcast = 1 | messages;  
end  
$fdisplay (broadcast,  
    "file opened at %d", $time);  
$fdisplay (messages,  
    "Error on r_color at %d",  
    $time);  
```

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.)
Listing 20: Queue

$q_{\text{initialize}} (q_{\text{id}}, q_{\text{type}}, \text{max\_length}, \text{status});
$q_{\text{add}} (q_{\text{id}}, \text{job}_{\text{id}}, \text{inform}_{\text{id}}, \text{status});
$q_{\text{remove}} (q_{\text{id}}, \text{job}_{\text{id}}, \text{inform}_{\text{id}}, \text{status});
$q_{\text{full}} (q_{\text{id}}, \text{status});
$q_{\text{exam}} (q_{\text{id}}, q_{\text{stat\_code}}, q_{\text{stat\_value}}, \text{status});
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions

Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 21: Sequential
always @(posedge clock) begin
    a = b;
c = d;
e = f;
g = h;
...
y = z;
end //always
always @(posedge clock) begin
    if (reset) state <= 1;
    else begin
        state <= state + 1;
    end
end //always

    //always
    case (state) begin
        1: begin
            a <= b;
            end
        2: begin
            c <= d;
            end
        ... 
        N: begin
            y <= z;
            end
        endcase
    end //always
for (t=0; t<T; t=t+1)
begin
  a = b;
  c = d;
  ...
  y = z;
end

while (t<T)
begin
  t = t + 1;
  a = b;
  ...
  y = z;
end

repeat (T)
begin
  a = b;
  c = d;
  ...
  y = z;
end
Loop 1

```verilog
always @ (posedge clock) begin
    if (reset) begin
        state <= 1; t <= 1;
    end else begin
        state <= state + 1;
        case (state) begin
            1: a <= b;
            2: c <= d;
            ...
            N: begin
                y <= z;
                if (t < T) begin
                    state <= 1;
                    t <= t + 1;
                end
            end
        endcase
    end
end //always
```

Loop 2

```verilog
always @ (posedge clock) begin
    if (reset) begin
        state <= 1; t <= T;
    end else begin
        state <= state + 1;
        case (state) begin
            1: a <= b;
            2: c <= d;
            ...
            N: begin
                y <= z;
                if (t > 0) begin
                    state <= 1;
                    t <= t - 1;
                end
            end
        endcase
    end
end //always
```
Autoincrement

always @ (posedge clock) begin
  if (reset) begin
    state <= 1; t <= 1;
  end else if (t < T) begin
    state <= state + 1;
    t <= t + 1;
    case (state) begin
      1: a <= b;
      2: c <= d;
      ...
      N: y <= z;
    endcase
  end
end //always

Autodecrement

always @ (posedge clock) begin
  if (reset) begin
    state <= 1; t <= T;
  end else if (t > 0) begin
    state <= state + 1;
    t <= t - 1;
    case (state) begin
      1: a <= b;
      2: c <= d;
      ...
      N: y <= z;
    endcase
  end
end //always
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes

Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Figure: The three-step design method for vision architecture.
\[
\begin{align*}
q(t + 1) &= T(q(t), x(t)), \\
y(t) &= H(q(t)).
\end{align*}
\]
\[ q(t+1) = T(q(t), x(t)) \]
\[ y(t) = H(q(t)) \]

\[ x(t) \]
\[ q(t) \]
\[ y(t) \]

(a) State machine

\[ x \]
\[ q(p) \]
\[ y \]
\[ q(p') \]

(b) RAM memory

\[ x \]
\[ q(p) \]
\[ y \]
\[ q(p') \]

(c) Queue

\[ x \]
\[ q(p) \]
\[ y \]
\[ q(p') \]

(d) Stack

Figure: State machines for vision.
\[
\begin{aligned}
q(p') & \leftarrow T(q(p), x), \\
y & \leftarrow H(q(p)).
\end{aligned}
\]
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture

Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 22: Referencing method: active sender and passive receiver

```vhdl
'define DATA_WIDTH 8
'define DATA_SIZE 10
module sender(input clock, reset);
    reg ['DATA_WIDTH-1:0] image [0:'DATA_SIZE-1]; //image to be sent
    integer i; //counter

//send data
    always @(posedge clock) begin
        if (reset) begin //pseudo data
            for (i=0; i<'DATA_SIZE; i=i+1) begin //by random
                image[i] <= $random % 256; //number generation
            end
        end
        else begin //copy data by name reference
            for (i=0; i<'DATA_SIZE; i= i+1) begin //send data
                RECEIVER.image[i] <= image[i]; //by referencing
            end
        end
    end //always
endmodule
```
module receiver(input clock, reset);
    reg ['DATA_WIDTH-1:0] image [0:'DATA_SIZE-1]; //image to be copied
endmodule
Listing 23: Referencing method: passive sender and active receiver

```vhdl
'define DATA_WIDTH 8
'define DATA_SIZE 10
module sender(input clock, reset);
    reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE-1]; //image to be sent
    integer i; //counter
    //send data
    always @(posedge clock) begin
        if (reset) begin //pseudo data
            for (i=0; i<‘DATA_SIZE; i=i+1) begin //by random
                image[i] <= $random % 256; //number generation
            end
        end
    end //always
endmodule

module receiver(input clock, reset);
    reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE-1]; //empty image
    integer i; //counter
    //copy data
    always @(posedge clock) begin
        //copy data by name reference
    end
endmodule
```

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.) Architectures for Computer Vision
for (i=0; i<DATA_SIZE; i= i+1) begin //copy data
    image[i] <= SENDER.image[i]; //by referencing
end
end //always
Listing 24: Referencing method: active third module

```verilog
`define DATA_WIDTH 8
`define DATA_SIZE 10
module sender(input clock, reset);
    reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE-1]; //image to be sent
    integer i; //counter
    //send data
    always @(posedge clock) begin
        if (reset) begin //pseudo data
            for (i=0; i<‘DATA_SIZE; i=i+1) begin //by random
                image[i] <= $random % 256; //number generation
            end
        end
    end //always
endmodule

module receiver(input clock, reset);
    reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE-1]; //image to be copied
endmodule

module third_module(input clock, reset);
endmodule
```

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.)
/instantiation
sender SENDER (clock, reset);
receiver RECEIVER (clock, reset);

//control by a third module
integer i; //counter
always @(posedge clock) begin
    for (i=0; i< DATA_SIZE-1; i= i+1) begin //send data
        RECEIVER.image[i] <= SENDER.image[i]; //copy image
    end
end //always
endmodule
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 25: Synchronous communication: third module control

```verilog
`define DATA_WIDTH 8
`define DATA_SIZE 10
module third_module (input clock, reset);
    integer i; //counter
    wire [7:0] data; //connection
    reg state, req_send, req_receive; //control variables

//instantiation
    sender SENDER (clock, reset, req_send, data); //sender
    receiver RECEIVER (clock, reset, req_receive, data); //receiver

//control
    always @(posedge clock) begin
        if (reset) begin
            i <= 0; state <= 1; req_send <= 0; req_receive <= 0;
        end
        else begin
            case (state)
                0: begin
                    state <= 1; i <= i + 1; req_send <= 1; req_receive <= 0;
                end
                1: if (i < 10) begin //trigger both modules
            end
            endcase
        end
    end
endmodule
```
state <= 1; i = i + 1; req_send <= 1; req_receive <= 1; end
else if (i == 10) begin //trigger once more
    i <= i + 1; req_send <= 0; req_receive <= 1;
end
else req_receive <= 0;
default: state <= 0;
endcase
endendmodule

module sender(input clock, reset, req, output reg [7:0] data);
    reg [7:0] image [0:9]; //image data
    //fill the image
    integer i; //counter

    //fill the image
    always @(posedge clock)
    if (reset) begin //provide data
        for (i=0; i<10; i=i+1) begin image[i] <= $random % 256;
    end

    always @(posedge clock) case (req) //monitor req_send

0: i <= 0;
1: begin i <= i + 1; data <= image[i]; end
endcase
endmodule

module receiver(input clock, reset, req, input [7:0] data);
  reg [7:0] image [0:9]; //empty image
  //sending data
  integer i; //counter
  always @(posedge clock) case (req) //sense req_receive
    0: i <= 0;
    1: begin i <= i + 1; image[i] <= data; end
  endcase
endmodule
Listing 26: Sender active receiver passive

`define DATA_WIDTH 8
`define DATA_SIZE 10
module sender(input clock, reset, output reg req, output reg [7:0] data);

  // image
  reg [7:0] image [0:9]; // image
  // variables
  integer i; // counter
  reg state; // state

  // fill the image
  always @(posedge clock) begin
    if (reset) begin
      for (i=0; i<10; i=i+1) image[i] <= $random % 256;
    end
  end // always

  // send data
  always @(posedge clock) begin
    if (reset) begin
      state <= 0; i <= 0;
    end
    else begin
      case (state)
        0: begin
          state <= 1; i <= i + 1; data <= image[i]; req <= 1;
      endcase
    end
  end // always
end

1: if (i < 10) begin
    i <= i + 1; data <= image[i]; req <= 1;
end
else if (i == 10) begin i <= i + 1; req <= 1; end
else req <= 0;
endcase
end
end //always
endmodule

module receiver (input clock, reset, req, input [7:0] data); //receiver
    //empty image
    reg [7:0] image [0:‘DATA_SIZE-1]; //empty image
    //variables
    reg state; //state
    integer i; //counter
    //main part
    always @(posedge clock) begin
        case (req) //monitor request
            0: i <= 0; //idle
            1: if (i < 10) begin i <= i + 1; image[i] <= data; end
                else i <= 0;
        endcase
    end
endmodule
end //always
endmodule
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 27: Handshaking

```verilog
'define DATA_WIDTH 8  
'define DATA_SIZE 10  
module third_module (input clock, reset);    //call the modules  
    //variables  
    wire ['DATA_WIDTH-1:0] data;                 //connecting the modules  
    //instantiation  
    sender SENDER (clock,reset,req,ack,data);  //sender  
    receiver RECEIVER (clock,reset,req,ack,data); //receiver  
endmodule

module sender(input clock,reset,output reg req,input ack,  
    output reg [7:0] data);                    //image  
    reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE-1]; //image for transfer  
    //variables  
    integer i;                                 //counter  
    reg state;                                 //state  
    //main part for sending  
    always @(posedge clock) begin              //fill the image  
        if (reset) begin                        
            for (i=0; i<10; i=i+1) image[i] <= $random % 256;  
        end
    end
```

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.)

Architectures for Computer Vision
Asynchronous Port Communication

```verilog
module receiver (input clock, reset, input req, output reg ack, 
    input [7:0] data);
    //empty image
    reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE-1]; //to be filled
```
//variables
reg state; //state
integer i; //counter
//main part for sending
always @(posedge clock) begin
  if (reset) begin i <= 0; state <= 1; ack <= 0; end
  else begin
    case (state)
      0: if (!req) begin state <= 1; ack <= 0; end
      1: if (req) begin
          if (i < 10) begin
            state <= 0; i <= i + 1; image[i] <= data; ack <= 1;
          end
          else begin state <= 0; i <= 0; ack <= 0;
          end
        end
    default: state <= 0;
    endcase
  end //always
endmodule
Listing 28: Port communication: Four phase handshake

```
'define DATA_WIDTH 8
'define DATA_SIZE 100

module tb; //testbench
    reg clock, reset;
    wire ['DATA_WIDTH - 1:0] data; //connection
    wire send, receive; //semaphores
    integer i; //counter

    //instantiation
    sender SENDER (clock, reset, send, receive, data); //sender
    receiver RECEIVER (clock, reset, send, receive, data); //receiver

    //initialization
    initial begin
        clock = 0; reset = 0; //initialize clock and reset
        for(i=0;i<'DATA_SIZE;i=i+1) SENDER.image[i] = $random % 256;
        #50; reset = 1; #150; reset = 0; //reset
    end
    //clock generation
    always #50 clock = ~clock;
```
module sender ( //sender
    input clock, reset,
    input send, //receive message from receiver
    output reg receive, //send message to receiver
    output reg ['DATA_WIDTH-1:0] data
);

reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE - 1]; //data

integer i; //variable

always @(posedge clock) begin: SENDER
    if (reset) begin i <= 0; end //initialize variable
    else if (i < ‘DATA_SIZE) begin
        receive = 0; //don’t receive yet
        wait (send) data = image[i]; //wait for send
        receive = 1; //receive my data
        wait (!send) receive = 0; //wait while receiving
        i = i + 1; //next data
    end
end //always
endmodule
module receiver (  
    input clock, reset,  
    output reg send,  
    input receive,  
    input ['DATA_WIDTH-1:0] data  
);

reg ['DATA_WIDTH-1:0] image [0:9];  
integer i;

always @ (posedge clock) begin: RECEIVER  
    if (reset) begin i = 0; end //initialize variable  
    else begin  
        send = 1; //send data  
        wait (receive) image[i] = data; //wait for receive  
        send = 0; //don’t send  
        wait (!receive) send = 1; //wait  
        i = i + 1; //next data  
    end //else  
end //always  
endmodule
Listing 29: Port communication: two-phase handshake

```
`define DATA_WIDTH 8
`define DATA_SIZE 100

module tb; //testbench
    reg clock, reset;
    wire [`DATA_WIDTH - 1:0] data; //connection
    wire send, receive; //semaphores
    integer i; //counter

    //instantiation
    sender SENDER (clock, reset, send, receive, data); //sender
    receiver RECEIVER (clock, reset, send, receive, data); //receiver

    //initialization
    initial begin
        clock = 0; reset = 0; //initialize clock and reset
        for(i=0;i<`DATA_SIZE;i=i+1) SENDER.image[i] = $random % 256;
        #50; reset = 1; #150; reset = 0; //reset
    end
    //clock generation
    always #50 clock = ~clock;
```
Asynchronous Port Communication

endmodule

module sender ( //sender
    input clock, reset,
    input send, //receive message from receiver
    output reg receive, //send message to receiver
    output reg ['DATA_WIDTH-1:0] data
);

reg ['DATA_WIDTH-1:0] image [0:‘DATA_SIZE - 1]; //data

integer i; //variable

always @(posedge clock) begin: SENDER
    if (reset) begin receive = 0;i <= 0;end //initialize variable
    else if (i < ‘DATA_SIZE) begin
        receive = ~receive; //don’t receive yet
        wait (send) data = image[i]; //wait for send
        i = i + 1; //next data
    end
end //always
endmodule

module receiver (
input clock, reset, 
output reg send, 
input receive, 
input [‘DATA_WIDTH-1:0] data 
);

reg [‘DATA_WIDTH-1:0] image [0:9];
integer i;

always @ (posedge clock) begin: RECEIVER
   if (reset) begin send = 0; i = 0; end  //initialize variable
   else begin
      send = ~send;  //send data
      wait (receive) image[i] = data;  //wait for receive
   end  //else
end  //always
endmodule
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 30: Communication: flattening the data

`define WIDTH 8
`define SIZE 10

module tb;
  reg clock, reset;

  initial begin
    clock = 0; reset = 0; #100; //initialize
    reset = 1; #100; reset = 0; //reset
  end

  always #50 clock = ~clock; //clock

  //fill the memory for test
  integer j;
  always @(posedge clock) begin
    if (reset) for (j = 0; j < 'SIZE; j = j+1) //not for synthesis
      SENDER.image[j] <= $random % 256; //use name reference
  end //always

  //main part for data connection
  wire ['WIDTH * 'SIZE - 1:0] data; //very big wire
sender SENDER (clock, reset, data); //sender
receiver RECEIVER (clock, reset, data); //receiver
endmodule

module sender( //sender
    input clock, reset,
    output ['WIDTH * 'SIZE - 1:0] data
);

reg ['WIDTH - 1:0] image [0:'SIZE -1]; //image to be sent

//make a single big data
genvar i;
genenerate //use generate
    for (i = 0; i < 'SIZE; i= i + 1) begin: GEN //fill the fields
        assign data['WIDTH*i +:'WIDTH] = image[i];
    end
endgenerate
endmodule

module receiver( //receiver
    input clock, reset,
    input ['WIDTH * 'SIZE - 1:0] data //big bus
);
reg ['WIDTH - 1:0] image [0:9]; //image data

//disassemble the data
integer i;
always @*
  for (i = 0; i < 'SIZE; i = i + 1)
    image[i] <= data['WIDTHG*i +:'WIDTH]; //original size
endmodule
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
Listing 31: Control: FSM

module tb; //testbench
  reg clock, reset;
  wire [2:0] q; //connection

    //instantiation
    control CONTROL (clock, reset, q, ackA, ackB, ackC); //control module
    moduleA A (clock, reset, q, ackA); //module A
    moduleB B (clock, reset, q, ackB); //module B
    moduleC C (clock, reset, q, ackC); //module C

    //clock and reset
    initial begin
      clock = 0; reset = 0; #50; //initialize
      reset = 1; #50; reset = 0; //reset signal
      forever #50 clock = ~clock; //clock signal
    end
endmodule

    //control unit
    module control ( //control unit
      input clock, reset,
output reg [2:0] q, //control signal
input ackA, ackB, ackC //status
);

//build input status vector
wire [2:0] qi;
assign qi[0] = ackA;
assign qi[1] = ackB;
assign qi[2] = ackC;

//determine next state
always @ (negedge clock, posedge reset) begin //negedge used
  if (reset) begin q <= 3'b000; end
  else case (q) //moore machine
    3'b000: q <= 3'b001;
    3'b001: if (qi == 3'b001) q <= 3'b010; //next state
    3'b010: if (qi == 3'b010) q <= 3'b100;
    3'b100: if (qi == 3'b100) q <= 3'b000;
    default: q <= 3'b000;
  endcase
end //always
endmodule

//datapath modules
module moduleA (input clock, reset, input[2:0] q, output reg ack); //a
  always @ (posedge clock, posedge reset) begin //posedge used
    if (reset) ack <= 0; //reset the status
    else if (q[0]) begin //monitor the control
      $display("A"); //arbitrary statements
      ack <= 1; //assert the status
    end else ack <= 0; //reset the status
  end //always
endmodule

module moduleB (input clock, reset, input[2:0] q, output reg ack); //b
  always @ (posedge clock, posedge reset) begin
    if (reset) ack <= 0;
    else if (q[1]) begin
      $display("B"); //arbitrary statements
      ack <= 1;
    end else ack <= 0;
  end //always
endmodule

module moduleC (input clock, reset, input[2:0] q, output reg ack); //c
  always @ (posedge clock, posedge reset) begin
    if (reset) ack <= 0;
    else if (q[2]) begin

Hong Jeong (©2014 John Wiley & Sons Singapore Pte Ltd.) Architectures for Computer Vision
$display("C");
ack <= 1;
end else ack <= 0;
end //always
endmodule
Listing 32: Control: distributed

```
module tb; //testbench
    reg clock, reset;
    reg run; //triggering signal
    //instantiation
    moduleA A (clock, reset, run, ackA); //module A
    moduleB B (clock, reset, ackA, ackB); //module B
    moduleC C (clock, reset, ackB, ackC); //module C

    //clock and reset
    initial begin
        clock = 0; reset = 0; run = 0;#50; //initialize
        reset = 1; #50; reset = 0; //reset signal
        run = 1; #100; run = 0; //run signal
    end
    always #50 clock = ~clock; //clock signal
endmodule

module moduleA (input clock, reset, input run, output reg ack);//a
    always @ (posedge clock, posedge reset) begin
        if (reset) ack <= 0; //reset the status
```
```verbatim
else if (run) begin //monitor the control
    $display("A"); //arbitrary statements
    ack <= 1; //assert the status
    end else ack <= 0;
end //always
endmodule

module moduleB (input clock, reset, input run, output reg ack);//b
    always @ (posedge clock, posedge reset) begin
        if (reset) ack <= 0; //reset the status
        else if (run) begin //monitor the control
            $display("B"); //arbitrary statements
            ack <= 1; //assert the status
        end else ack <= 0; //reset the status
    end //always
endmodule

module moduleC (input clock, reset, input run, output reg ack);//c
    always @ (posedge clock, posedge reset) begin
        if (reset) ack <= 0; //reset the status
        else if (run) begin //monitor the control
            $display("C"); //arbitrary statements
            ack <= 1; //assert the status
        end else ack <= 0; //reset the status
    end //always
endmodule
```
end //always
endmodule
Contents

The Verilog System
Hello, World!
Modules and Ports
UUT and TB
Data Types and Operations
Assignments
Structural-Behavioral Design Elements
Tasks and Functions
Syntax Summary
Simulation-Synthesis
Verilog System Tasks and Functions
Converting Vision Algorithms into Verilog HDL Codes
Design Method for Vision Architecture
Communication by Name Reference
Synchronous Port Communication
Contents (cont.)

Asynchronous Port Communication

Packing and Unpacking

Module Control

Procedural Block Control
module tb;
reg clock, reset;
reg [1:0] state;

initial begin
  clock = 0; reset = 0; #50; reset = 1; #50; reset = 0;
  end
always #50 clock = ~clock;

always @(posedge clock) begin
  if (reset) begin
    state <= 0;
  end
  else case (state)
    0: begin
      state <= 1;
      $display("A");
    end
    1: begin
      state <= 2;
      $display("B");
    end
  endcase
end
```plaintext
end
2: begin
state <= 0;
$display("C");
end
default: state <= 0;
endcase
end
endmodule
```
Listing 34: Controlling procedural blocks: semaphores

module tb1;

reg clock, reset;
reg [1:0] state_A, state_B, state_C;
reg do_A, do_B, do_C;

initial begin
    clock = 0; reset = 0; #50; reset = 1; #50; reset = 0;
    end
always #50 clock = ~clock;

always @(posedge clock) begin: BLOCK_A
    if (reset) begin
        state_A <= 0;
        do_B <= 0;
        end
    else case (state_A)
        0: begin
            if (do_A) begin
                state_A <= 1;
                do_C <= 0;
            end
    end
end
else begin
state_A <= 0;
do_B <= 0;
end
end
1: begin
state_A <= 0;
$display("A");
do_B <= 1;
end
default: state_A <= 0;
endcase
end

always @(posedge clock) begin: BLOCK_B
if (reset) begin
state_B <= 0;
do_C <= 0;
end
else case (state_B)
0: begin
if (do_B) begin

Hong Jeong ([©2014 John Wiley & Sons Singapore Pte Ltd.)
state_B <= 1;
do_C <= 0;
end
else begin
state_B <= 0;
do_C <= 0;
end
end
1: begin
state_B <= 0;
$display("B");
do_C <= 1;
end
default: state_B <= 0;
endcase
end

always @(posedge clock) begin: BLOCK_C
if (reset) begin
state_C <= 0;
do_A <= 1;
end
else case (state_C)
0: begin
if (do_C) begin
  state_C <= 1;
  do_B <= 0;
  end
else begin
  state_C <= 0;
  do_A <= 0;
  end
end
1: begin
  state_C <= 0;
  $display("C");
  do_A <= 1;
  end
default: state_C <= 0;
endcase
end
endmodule
Listing 35: Controlling procedural blocks: control unit

module tb;
reg clock, reset;
reg [1:0] state, state_A, state_B, state_C;
reg do_A, do_B, do_C, done_A, done_B, done_C;

initial begin
  clock = 0; reset = 0; #50; reset = 1; #50; reset = 0;
end
always #50 clock = ~clock;

always @(posedge clock) begin
  if (reset) begin
    state <= 0;
    do_A <= 1;
    do_B <= 0;
    do_C <= 0;
  end
  else case (state)
    0: if (done_A) begin
      do_A <= 0;
      do_B <= 0;
      do_C <= 0;
    end
  endcase
end
do_B <= 1;
state <= 1;
end
1: if (done_B) begin
do_B <= 0;
do_C <= 1;
state <= 2;
end
2: if (done_C) begin
do_C <= 0;
do_A <= 1;
state <= 0;
end
default: state <= 0;
endcase
end

always @(posedge clock) begin: BLOCK_A
if (reset) begin
done_A <= 0;
end
else if (do_A) begin
$display("A");
done_A <= 1;
end
end
else
done_A <= 0;
end

always @(posedge clock) begin: BLOCK_B
if (reset) begin
done_B <= 0;
end
else if (do_B) begin
$display("B");
done_B <= 1;
end
else
done_B <= 0;
end

always @(posedge clock) begin: BLOCK_C
if (reset) begin
done_C <= 0;
end
else if (do_C) begin
$display("C");
done_C <= 1;
end
else
done_C <= 0;
end
endmodule
Listing 36: Control: internal trigger

module tb; //testbench
    reg clock, reset;
    reg run; //first module trigger

    //instantiation
    moduleABC ABC (clock, reset, run); //moduleABC

    //clock and reset
    initial begin
        clock = 0; reset = 0; run = 0; //clock, reset, run
        #50; reset = 1; #50; reset = 0; //reset signal
        run = 1; #100; run = 0; //run signal
    end
    always #50 clock = ~clock;
endmodule

module moduleABC (input clock, reset, input run);
    reg ack, ackB, ackC;
    always @ (posedge clock, posedge reset) begin: A
        if (reset) ackB <= 0; //reset the status
        else @ (run) begin //monitor the trigger
            if (run) ack <= 1
            else ack <= 0
        end
    end
endmodule
``` VERILOG
$display("A"); //arbitrary statements
ackB <= 1;
end
end

always @(posedge clock, posedge reset) begin: B
  if (reset) ack <= 0; //reset the status
  else @(ackB) begin //monitor the trigger
    $display("B"); //arbitrary statements
    ackC <= 1;
  end
end

always @(posedge clock, posedge reset) begin: C
  if (reset) ack <= 0; //reset the status
  else @(ackC) begin //monitor the trigger
    $display("C");
  end
end
endmodule
```