ECE 4170
Introduction to HDLs with Applications to Digital Systems

Introduction to Verilog

Disclaimer: lecture notes based on originals by Giovanni De Micheli

History

- Developed at Gateway in '83.
  - Proprietary language.
  - Support for simulation and testing tools.
- Cadence bought Gateway.
  - The language was open to the public.
- Open Verilog international:
  - Promote the use of the language.

Language goal

© GDM

- Hardware specification and simulation.
  - Abstraction levels:
    - Function, logic, switch.
  - Views:
    - Behavior, structure.
- Support iterative refinment of a model.

General features

© GDM

- Module *definition* and *instantiation*.
- Hierarchy.
- Mixed-mode modeling.
- Strongly tied to simulator.
- Overall flavor:
  - Simple and terse.
  - C-like look and feel.

Modules

- Ports:
  - *inputs*, *outputs* and *inouts*.
- Data types:
  - *net* and *register*.
- Assignments:
  - *Continuous* assignments.
  - *Procedural* assignments.

Data types

- Related to electrical properties and simulation requirements.
- *net*:
  - *wire*: modeling connection (memoryless).
  - *wor*, *wand*: wired `OR`, `AND` for ECL.
  - *trireg* : net with capacitive load (memory).
- *reg*:
  - registers (store values).
- Additional minor types: e.g., *supply* ...

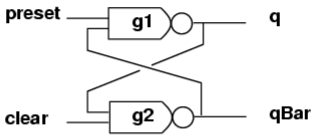
Assignments

- Continuous assignments.
  - Syntax: *assign* $x = a + b$;
  - Executes always on event on r.h.s.
  - Applies to wires.
- Procedural assignments.
  - Syntax: $x = a + b$;
  - Applies to registers.
  - Executes when reached in model execution.

Views

- Structural modeling.
  - Define interconnection of modules.
  - Use *continuous assignment* only.
- Behavioral modeling.
  - Procedural semantics.
  - *Continuous* and *procedural* assignments.
  - Control-flow constructs.

Example of structural modeling
latch



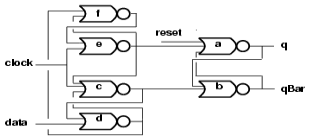

```
module ffNand;
    wire q, qBar;
    reg  preset, clear;

    nand #1
        g1 (q, qBar, preset),
        g2 (qBar, q, clear);

endmodule
```

Example of simulation
latch


```
//Example 1.2.  NAND Latch To Be Simulated.
module ffNand;
    wire q, qBar;
    reg  preset, clear;

    nand #1
        g1 (q, qBar, preset),
        g2 (qBar, q, clear);

    initial
        begin
            // two slashes introduce a single line comment
            $monitor ($time,,
                "Preset = %b clear = %b q = %b qBar = %b",
                preset, clear, q, qBar);
            //waveform for simulating the nand flip flop
            #10 preset = 0; clear = 1;
            #10 preset = 1;
            #10 clear = 0;
            #10 clear = 1;
            #10 $finish;
        end
endmodule
```

Example of structural modeling
D-type Flip-Flop

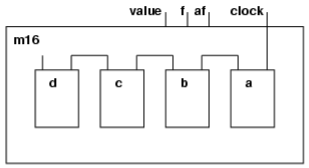
© GDM




```
module dEdgeFF (q, clock, data);
    output q;
    reg    reset;
    input  clock, data;
    wire   q,qBar,r,s,r1,s1;

    initial begin
        reset = 1;
        #10 reset =0;
    end

    nor #10
        a(q,qBar,r,reset);
    nor
        b(qBar,q,s),
        c(s,r,clock,s1),
        d(s1,s,data),
        e(e,r1,clock),
        f(r1,s1,r);
endmodule
```

Example of hierarchy

© GDM




```
//Example 1.4.  A 16-Bit Counter.
module m16 (value, clock, f, aF);
    output [3:0] value;
    output       f;
                 aF;
    input        clock;

    dEdgeFF    a (value[0], clock, ~value[0]),
               b (value[1], clock, value[1] ^  value[0]),
               c (value[2], clock, value[2] ^ &value[1:0]),
               d (value[3], clock, value[3] ^ &value[2:0]);

    assign f = value[0] & value[1] & value[2] & value[3];
    assign aF = &value;
endmodule
```

Behavioral modeling

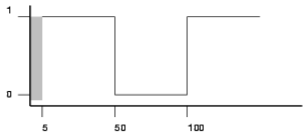
- Set of procedural statements.
- Process repeats itself:
  - *always*.
- Initialization:
  - *initial*.
- Event synchronization: @
- Time delay #

Example of behavioral modeling
D-type Flip-Flop


```
module dEdgeFF (q, clock, date);
    output q;
    reg    q;
    input clock, data;

    initial
        q=0

    always
        @(negedge clock) #10 q = data;

endmodule
```

Example
m555

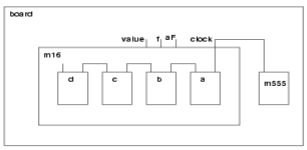
© GDM




```
module m555 (clock);
   output   clock;
   reg      clock;

   initial
     #5 clock = 1;

   always
     #50 clock = ~ clock;
endmodule
```

Example
board

© GDM




```
module board;
   wire    [3:0]    count;
   wire             count,
                    f,
                    af;

   m16     counter (count, clock, f, af);
   m555    clockGen (clock);

   always @ (posedge clock)
      $display ($time,,,"count%d,f=%d",af=%d",
                                    count,f,af);
endmodule
```

Example simulation


```
5 count=x,f=x,af=x
100 count=1,f=0,af=0
200 count=2,f=0,af=0
300 count=3,f=0,af=0
400 count=4,f=0,af=0
500 count=5,f=0,af=0
600 count=6,f=0,af=0
700 count=7,f=0,af=0
800 count=8,f=0,af=0
900 count=9,f=0,af=0
1000 count=10,f=0,af=0
1100 count=11,f=0,af=0
1200 count=12,f=0,af=0
1300 count=13,f=0,af=0
1400 count=14,f=0,af=0
1500 count=15,f=1,af=1
1600 count=0,f=0,af=0
1700 count=1,f=0,af=0
1800 count=2,f=0,af=0
1900 count=3,f=0,af=0
```

Behavioral modeling constructs

- *always*.
- *initial*.
- *if then else*.
- *if else if*.
- *case*.
- *repeat, while, for, forever*.

Example
mark1


```
module mark1Case;
   reg[31:0] m[0:8191];      //8192 x 32 bit memory
   reg[12:0] pc;             //13 bit program counter
   reg[31:0] acc;            //32 bit accumulator
   reg[15:0] ir;             //16 bit instruction register

   always
     begin
       ir = m[pc];
       case (ir[15:13])
        3'b000: pc = m[ir[12:0]];
        3'b001: pc = pc + m[ir[12:0]];
        3'b010: acc = -m[ir[12:0]];
        3'b011: m[ir[12:0]] = acc;
        3'b100,
        3'b101: acc = acc - m[ir[12:0]];
        3'b110: if(acc<0)pc=pc+1;
       endcase
       pc=pc+1;
   end
endmodule
```

Logic values and consequences

- Values: *1* , *0* , *X* , *Z* .
- Ambiguity in conditionals:
  - *caseZ*:
    - Z is a *don't care*.
  - *caseX*:
    - X and Z are *don't cares*.

Tasks and Functions

- Used in behavioral modeling
- *task*:
  - equivalent to a software procedure
- *function*:
  - equivalent to a software function

Tasks and Functions

- A function executes in a time unit
- A task can contain time-controlling statements
- A function cannot enable a task
- A task can enable other tasks and functions
- A function must have at least one input argument
- A task may have zero or more arguments
- A function returns a single value
- A task does not return any value

Example of Task


```
module mark1Task;
   //declarations as before

   always
     begin
       ir=m[pc];
       case(ir[15:13]);
         // other case expressions as before
         3'b111: multiply(acc, m [ir [12:0]]);
       endcase
       pc = pc + 1;
     end

   task multipy;
     inout [31:0] a;
     input [31:0] b;

     reg [15:0]  mcnd, mpy;  //multiplicand and multiplier
     reg [31:0]  prod;       //product

     begin
       mpy =b[15:0];
       mcnd =a[15:0];
       prod =0;
       repeat(16)
         begin
           if (mpy[0])
                prod = prod + {mcnd,16'h0000};
           prod = prod>>1;
           mpy = mpy>>1;
         end
       a = prod;
     end
   endtask

endmodule
```

Example of Function


```
module mark1Fun;
   //declarations as before

   always
     begin
       ir=m[pc];
       case(ir[15:13]);
         //case expressions, as before
         3'b111: acc = multiply(acc, m [ir [12:0]]);
       endcase
       pc = pc + 1;
     end

   function [31:0] multiply;
     input [31:0] a;
     input [31:0] b;

     reg [15:0]  mcnd,mpy;

     begin
       mpy =b[15:0];
       mcnd =a[15:0];
       multiply = 0;
       repeat(16)
         begin
           if (mpy[0])
               multiply = multiply + {mcnd,16'h0000};
           multiply = multiply>>1;
           mpy = mpy>>1;
         end
       end
     endfunction

endmodule
```

Concurrent processes

- Set of concurrent modules.
- Synchronization events:
  - Receiving:
    - *@ (event) ; @ (posedge event) ; @ (expr);*
  - Triggering:
    - $\rightarrow$ *event*
- Hierarchical names: *processname.event*.

Example


```
module topNE();
  wire [15:0] number, numberOut;

  numberGenNE    ng(number);
  fibNumberGenNE fng(number, numberOut);
endmodule

module numberGenNE(number);
  output [15:0] number;
  reg    [15:0] number;
  event         ready

  initial
    number = 3;

  always
    begin
      #100 number=number + 1;
      -> ready;                  //generate event signal
    end
endmodule

module fibNumberGenNE(startingValue, fibNum);
  input  [15:0]  startingValue;
  output [15:0]  fibNum;

  reg  [15:0]    myValue;
  reg  [15:0]    fibNum;

  always
    begin
      @ng.ready                //accept event signal
        myValue = startingValue
      for (fibNum = 0; myValue !=0; myValue = myValue -1)
        fibNum = fibNum + myValue;
      $display ("%d, fibNum=%d", $time, fibNum);
    end
endmodule
```

Wait statement

© GDM

- Synchronization on a level:
  - *wait (expr);*
- Cannot be replaced by:
  - *while (expr)* {} ;
- Simulator-based semantics.

Simulation semantics

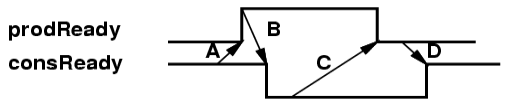
© GDM

- Simulator runs on uni-processor.
- Each process is simulated one at a time:
  - Input parameters are sampled and held.
  - On entrance of *always* portion:
    simulator runs until synchronization
    - E.g., external *wait*.
  - It then switches to other models.
- Waits are necessary to avoid endless loops in simulation.

Example

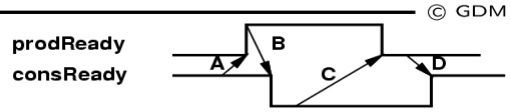

```
module endlessLoop (inputA);
input          inputA;
reg [15:0]     count;

    always
       begin
              count = 0;
              while (inputA)
                 count = count + 1;
                 //wait for inputA to go to 0
              $display ("This will never print");
       end

endmodule
```

Example
producer/consumer




```
module producer(dataOut, prodReady, consReady);
   output  [7.0]  dataOut;
   output         prodReady;
   input          consReady;

   reg            prodReady;
   reg     [7.0]  dataOut,
                  temp;

   always
     begin
       prodReady = 0;        //indicate nothing to transfer
       forever
         begin
           //...produce data and put into "temp:
           wait (consReady) // wait for consumer ready
             dataOut = $random;
           prodReady = 1;   //indicate ready to transfer
           wait (!consReady)        //finish handshake
             prodReady = 0;
         end
     end
endmodule
```

Example (2)




```
module consumer(dataIn, prodReady, consReady);
   input [7.0]  dataIn;
   input        prodReady;
   output       consReady;

   reg          consReady;
   reg   [7.0] dataInCopy;

   always
     begin
       consReady = 1;       //indicate consumer ready
       forever
         begin
           wait (prodReady)
              dataInCopy = dataIn;
           consReady = 0;   //indicate value consumer
           //...munch on data
           wait (!prodReady)         //complete handshake
              consReady = 1;
         end
      end
endmodule


module ProducerConsumer;
   wire [7.0] data;
   wire       pReady, cReady;

   producer   p(data, pReady, cReady);
   consumer   c(data, pReady, cReady);
endmodule
```

Disable breaks

- Asynchronous exit for loop.
- Asynchronous skip to next iteration in loops.
- Asynchronous exit from process.
  - It stops also called tasks and functions.

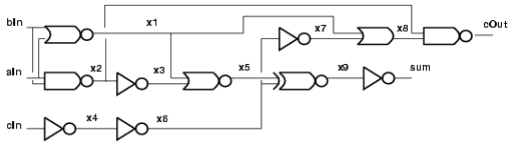
Example


```
module numberGenDisable (number, reset);
   output  [15:0] number;
   input          reset;
   event          ready;
   reg     [15:0] number;

   always
     begin :generator
           number = 3;
           forever
              begin
                #100 number=number + 1;
                -> ready;
              end
     end

   always
     @(negedge reset) disable generator;
endmodule
```

Logic level modeling

- Called also gate-level modeling.
- 12 gate-level primitives:
  - Examples: AND, OR, XOR.
- Structural and behavioral views.
  - Structural:
    - netlist.
  - Behavioral:
    - *always*, c-assignment, instantiation.

Example of structural view




```
module fullAdder(cOut, sum, aIn, bIn, cIn,);
   output cOut, sum;
   input  aIn, bIn, cIn;

   wire   x2;

   nand   (x2, aIn,bIn),
          (cOut, x2, x8);
   xnor   (x9, x5, x6);
   nor    (x5, x1, x3);
          (x1, aIn, bIn);
   or     (x8, x1, x7);
   not    (sum, x9),
          (x3, x2),
          (x6, x4),
          (x4, cIn),
          (x7, x6);
endmodule
```

Example of behavioral view


```
module oneBitFullAdder(cOut, sum, aIn, bIn, cIn);
  output cOut, sum;
  input  aIn, bIn, cIn;

  assign sum = aIn ^ bIn ^ cIn,
    cOut = (aIn & bIn) | (bIn & cIn) | (aIn & cIn)

endmodule
```

Gate level syntax

- *gate-type strength delay list-of-instances.*
- Four-level logic: {0,1,X,Z}.
- Strength:
  - Electrical drive.
- Wires:
  - Explicitly or implicitly defined.
- Ports:
  - Continuous assignment.

Strength specification

Strength Name	Strength Level	Element Modeled	Declaration Abbreviation	Printed Abbreviation
Supply Drive	7	Power supply connection	supply	Su
Strong Drive	6	Default gate and assign output strength	strong	St
Pull Drive	5	Gate and assign output strength	pull	Pu
Large Capacitor	4	Size of trireg net capacitor	large	La
Weak Drive	3	Gate and assign output strength	weak	We
Medium Capacitor	2	Size of trireg net capacitor	medium	Me
Small Capacitor	1	Size of trireg net capacitor	small	Sm
High Impedance	0	Not applicable	highz	Hi

Delay specification

- Applies to gates, nets and c-assignments.
- #(rise , fall , dtoZ).
- Extensible to:
  - Minimum, typical, maximum.
    - #(min-rise: typ-rise: max-rise , fall, dtoZ).
  - Block delays:
    - Delay paths through a block.

Example


```
module IOBuffer (bus, in, out, dir);
   inout  bus;
   input  in, dir;
   output out;

   parameter
     R_Min = 3, R_Typ = 4, R_Max = 5,
     F_Min = 3, F_Typ = 5, F_Max = 7,
     Z_Min = 12, Z_Typ = 15, Z_Max = 17;

   bufif1  #(R_Min: R_Typ: R_Max,
             F_Min: F_Typ: F_Max,
             Z_Min: Z_Typ: Z_Max)
           (bus, out, dir);

  buf      #(R_Min: R_Typ: R_Max,
             F_Min: F_Typ: F_Max,
           (in, bus);

endmodule
```

Example


```
module dEdgeFF (clock, d, clear, preset, q);
   input clock, d, clear, preset;
   output q;

   specify
     // specify parameters
     specparam  tRiseClkQ =100,
                tFallClkQ =120,
                tRiseCtlQ =50,
                tFallCtlQ =60,

    // module path declarations
    (clock => q) = (tRiseClkQ, tFallClkQ);
    (clear, preset *> q) = (tRiseCtlQ, tFallCtlQ);
  endspecify

   // description of module's internals
endmodule
```

User-defined logic primitives

- Defined by Boolean cover.
- Rules:
  - Multiple-input, single-output functions.
  - Output port is first on the list.
  - No *inouts* allowed.
  - No vectors.
  - Z input value not allowed.
- Extensible to level and edge sensitive sequential primitives.

Example

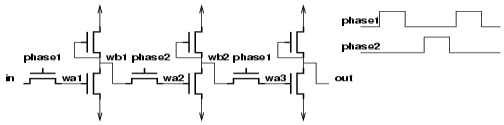

```
primitive carryX(carryOut, carryIn, aIn, bIn);
   output carryOut;
   input  aIn,
          bIn,
          carryIn;

   table
      0 00  :  0;
      0 01  :  0;
      0 10  :  0;
      0 11  :  1;
      1 00  :  0;
      1 01  :  1;
      1 10  :  1;
      1 11  :  1;
      0 0x  :  0;
      0 x0  :  0;
      x 00  :  0;
      1 1x  :  1;
      1 x1  :  1;
      x 11  :  1;
   endtable
endprimitive
```

Switch level modeling

- 14 switch-level primitives:
  - Examples: *nmos, pmos.*
- Structural view:
  - netlist.

Example

© GDM




```
//Example 1.2.  NAND Latch To Be Simulated.
module ffNand;
    wire q, qBar;
    reg  preset, clear;

    nand #1
        g1 (q, qBar, preset),
        g2 (qBar, q, clear);

    initial
        begin
            // two slashes introduce a single line comment
            $monitor ($time,,
                "Preset = %b clear = %b q = %b qBar = %b",
                preset, clear, q, qBar);
            //waveform for simulating the nand flip flop
            #10 preset = 0; clear = 1;
            #10 preset = 1;
            #10 clear = 0;
            #10 clear = 1;
            #10 $finish;
        end
endmodule
```

Summary

© GDM

- Mixed-mode modeling language:
  - Function, logic, switch.
- Tied to simulator.
- Support for stepwise design refinement.
- Widely distributed and used.