Introduction to Design With Verilog
Dr. Paul D. Franzon

Outline
1. HDL-based Design Flow
2. Introduction to the Verilog Hardware Description Language
3. A complete example: count.v
4. Basic Verilog features
5. Design flow

References
1. Quick Reference Guides
2. Smith & Franzon, Chapter 2-6
Attachments Required: Standard Synthesis Script (count.dc)
See Course Outline for further list of references
Role of Hardware Description Languages

Modern digital chip and system design centers on the use of Hardware Description Languages (HDL) to capture the design at the Register Transfer Level (RTL)

- RTL specifies all registers (flip-flops) and the combinational logic between the flip-flops
- Capturing the design in RTL is much faster than drawing a schematic
- Modern design depends heavily on the use of Computer-Aided Design tools:
  - To synthesize the RTL design into a schematic (or netlist)
  - To turn the schematic into a chip layout, FPGA mapping or board layout
  - To verify the original design, and verify that the more detailed designs are consistent with the original design
- Good designers depend critically on their ability to operate effectively with the CAD tools
  - Just knowing how to design logic is not enough
  - Unfortunately, you must learn a lot of tools and learn how to deal with their complexity and bugs
  - It’s important to form a good understanding of the tool’s methodology
ASIC Design Flow

- Develop C/Matlab/? Model
- Design Hardware
- Develop Timing Diagram
- Write Verilog Code

- High level *simulatable* design
  - Permits debugging of design intent
  - Reduces design cycle time
- Design **before** writing the Verilog code
  - HDLs speed up detailed gate design, not design capture
  - Design = hardware blocks + timing
Hardware Description Languages

- Verilog
  - Based on C, originally Cadence proprietary, now an IEEE Standard
  - Quicker to learn, read and design in than VHDL
  - Has more tools supporting its use than VHDL

- VHDL
  - VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
  - Developed by the Department of Defence, based on ADA
  - An IEEE Standard
  - More formal than Verilog
    - e.g. Strong typing
  - Has more features than Verilog
Verilog Model of a D-Flip Flop

module flipflop (D, clock, Q);
input D, clock;
output Q;
reg Q;
always@(posedge clock)
begin
  Q <= D;
end
endmodule

1. Module header, parameter list (=connected signals) and end
2. Declarations of parameter list
3. Local `variables`
   All assigned variables must be declared
4. Procedural block (or `sequential block`) forming `main body`
   `main body` can consist of several procedural blocks and other statements
VHDL Model of a D-Flip Flop

entity flipflop is
  port (clock, D: in bit;
       Q: out bit);
end flipflop;

architecture test of flipflop is
begin
  process
    begin
      wait until clock’event and clock = ‘1’;
      Q <= D;
    end process;
  end process;
end test;
Design Example .... Count Down Timer

Step 1: Write Specification:
- 4-bit counter
- count value loaded from 'in' on a positive clock edge when 'latch' is high
- count value decremented by 1 on a positive clock edge when 'dec' is high
- decrement stops at 0
- 'zero' flag active high whenever count value is 0

Simulatable Specification (C):

```c
for (value=in; value>=0; value--)
    if (value==0) zero = 1
    else zero = 0;
```

Produce Specification
- Simulatable
- Block (Module) Diagram
- All functions and wires
... Design Example

Step 2: Design the Hardware

- Clearly identify
  - All registers
  - Chunks of combinational logic
  - All internal signals
- Develop timing diagram

ALWAYS DESIGN THE HARDWARE BEFORE CODING

\[
\begin{align*}
\text{in} & \quad \text{latch} \\
\text{dec} & \quad \text{clock} \\
\end{align*}
\]
Step 3. Write the Verilog
module counter (clock, in, latch, dec, zero);
/* simple top down counter with zero flag */
input clock;  /* clock */
input [3:0] in;     /* starting count */
input latch;  /* latch `in' when high */
input dec;    /* decrement count when dec high */
output zero;   /* high when count down to zero */
reg [3:0] value;   /* current count value */
wire zero;
// register `value' and associated input logic
always@(posedge clock)
begin
  if (latch) value <= in;
  else if (dec && !zero) value <= #1 value - 1'b1;
  else value <= 4'h0;
end
// combinational logic to produce `zero' flag
assign zero = (value == 4'b0);
endmodule /* counter */
Features in Verilog Code

Note that it follows the hardware design, not the `C' specification

Multibit variables:

```
reg [3:0] value;
4-bit `'signal' [MSB:LSB]
```

Specifying constant values:

```
1'b1; 4'b0;
size 'base value; size = # bits, HERE: base = binary
NOTE: zero filled to left
```

Procedural Block:

```
always@(    ) Executes whenever variables in sensitivity list ( ) change value
begin change as indicated
end
```}

```
Usually statements execute in sequence, i.e. procedurally
begin ... end only needed if more than one statement in block
```
Design Example ... Verilog

- **Continuous Assignment:**
  assign is used to implement combinational logic directly

- **Questions**
  1. When is the procedural block following the always@(posedge clock) executed?
  2. When is the procedural block following the always@(latch or value or in or dec or zero) executed?
  3. How is a comment done?
  4. What does 1'b1 mean?
  5. What does reg [3:0] value; declare?
More Sophisticated Style

- Combine input logic to register with register

```verilog
always@(posedge clock)
begin
  if (latch) value = in;
  else if (dec && !zero) value = value - 1'b1;
  else value = 4'b0; // default is clear
end
assign zero = ~|value;
```
A Basic Verilog Style

Verilog is a powerful and flexible language
- It is very easy to describe functions that do NOT map well (or synthesize into) hardware
- It is also very easy to describe garbage that has no direct correlation to HW

Here is a suggested basic style for Verilog usage:
- For Registers and their associated input logic:
  ```verilog
  always@(posedge clock)
  begin
    if (case) register_output1 <= #1 register_input1;
    else register_output2 <= #1 register_input2;
  end
  ```
- For Selectors, Deselectors, etc:
  ```verilog
  always@(input1 or input2 or ...)
  begin
    if-then-else or case statement
  end
  ```
- For Simpler Combinational Logic:
  ```verilog
  assign output = inputs and operators
  ```
Summary so Far

- How do you build a flip-flop in Verilog?
- How does Verilog handle the intrinsic parallelism of hardware?
- What is a procedural block?
- What is continuous assignment?

always@(posedge clock)
Q <= D;