

Introduction to Design With Verilog

Dr. Paul D. Franzon

Outline

1. Introduction to the Verilog Hardware Description Language
2. A complete example: `count.v`

References

1. Franzon/Smith, Chapter 2-6
2. Sutherland, Quick Reference Guide

Hardware Description Languages

Verilog

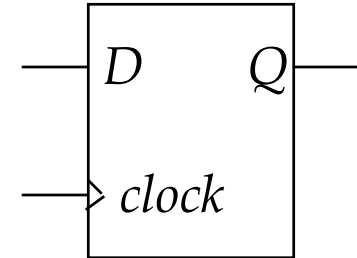
- Based on C, originally Cadence proprietary, now an IEEE Standard
- Quicker to learn, read and design in than VHDL
- Has more tools supporting its use than VHDL

VHDL

- VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
- Developed by the Department of Defense, based on ADA
- Now 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);①  
input D, clock;  
output Q;②  
reg Q;③  
always@(posedge clock)  
begin  
    Q <= D;④  
end  
endmodule①
```



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 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 test;
```

Design Example Count Down Timer

Produce Specification

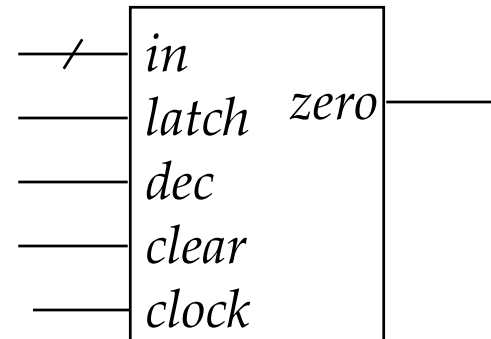
- *Simulatable*
- *Block (Module) Diagram*
- *All functions and wires*

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
- count value cleared when 'clear' is high
- decrement stops at 0
- 'zero' flag active high whenever count value is 0

Simulatable Specification (C):

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

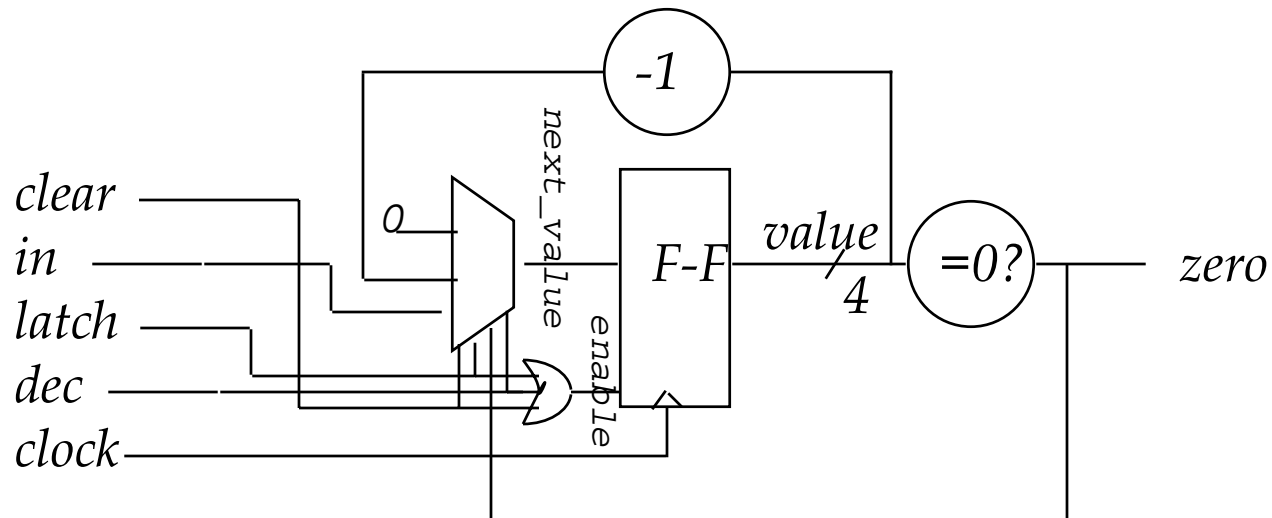


... Design Example

Step 2: Design the Hardware

- Clearly identify
 - ◆ All registers
 - ◆ Chunks of combinational logic
 - ◆ All internal signals

ALWAYS DESIGN THE HARDWARE BEFORE CODING



Step 3. Write the Verilog

```
module counter (clock, in, latch, dec, clear, 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 */
input          clear; /* clear count when clear high */
output         zero;  /* high when count down to zero */

reg [3:0] value; /* current count value */
reg [3:0] next_value;
wire      zero, enable;

// D-Flip Flops with enable
always@(posedge clock)
    if (enable) value <= next_value;

// produce enable
assign enable = latch | (dec & !zero) | clear;

// input multiplexor to value register
always@(latch or value or in or dec or zero)
    begin
        if (latch) next_value = in;
        else if (dec && !zero) next_value = value - 1'b1;
        else next_value = 4'b0; // default is clear
    end

// combinational logic to produce `zero' flag
    assign zero = ~|value;
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
  begin           value 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?

Simplified 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

You will be constrained to a subset of the language and a specific style of usage :

- For Registers:

```
always@(posedge clock)
begin
    register_output1 <= register_input1;
    register_output2 <= register_input2;
end
```

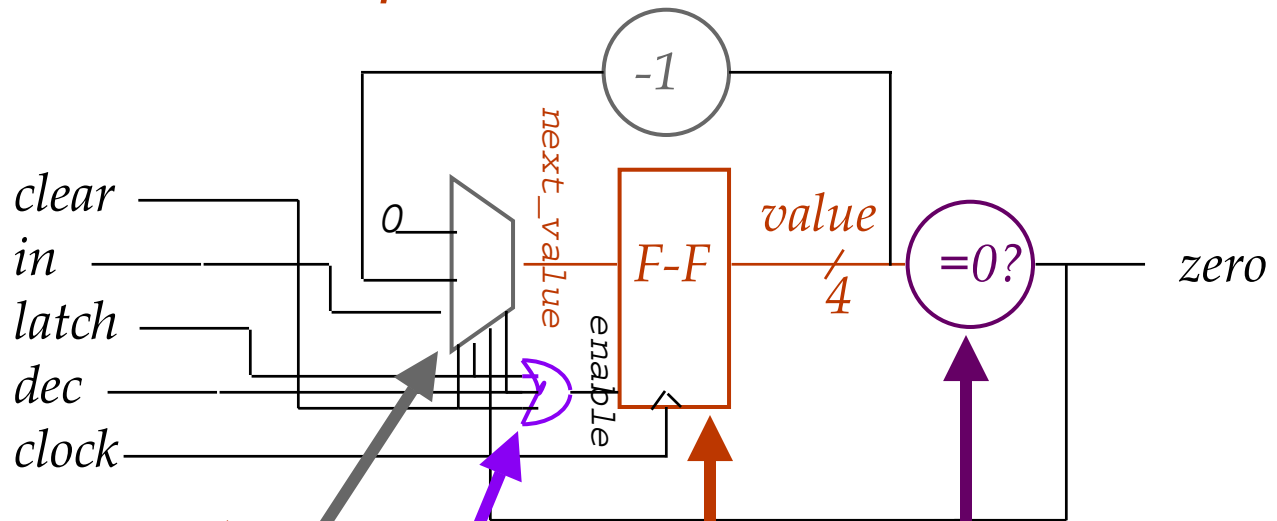
- For Selectors, Deselectors, etc:

```
always@(input1 or input2 or ...)
begin
    if-then-else or case statement
end
```

- For Simple Combinational Logic:

```
assign output = inputs and operators
```

Example



```

always@(posedge clock)
  if (enable) value <= next_value;

assign enable = latch | (dec & !zero) | clear;

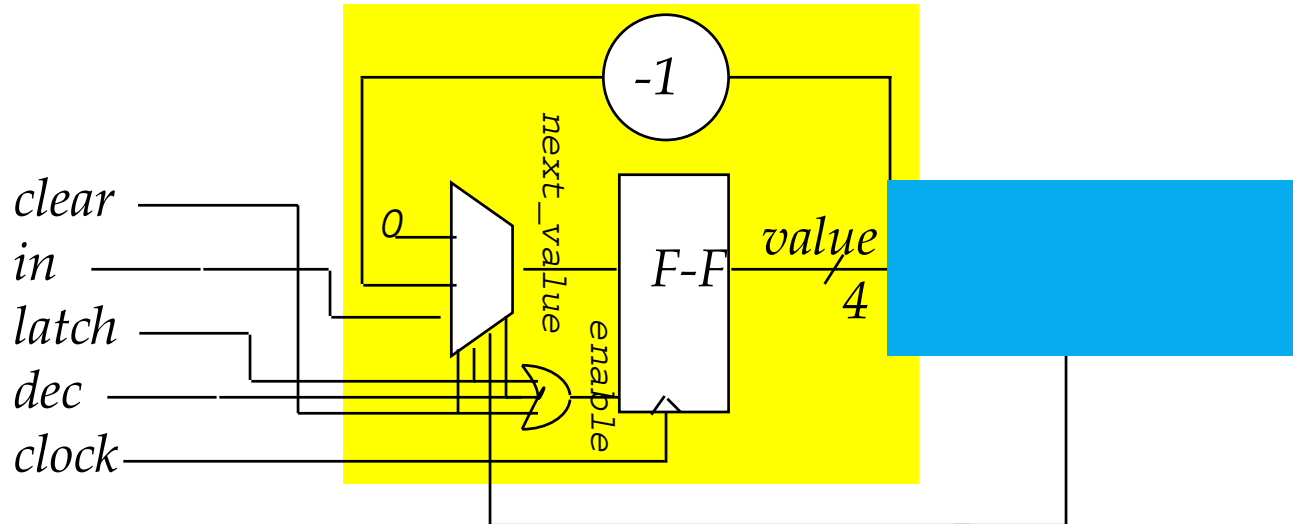
always@(latch or value or in or dec or zero)
  begin
    if (latch) next_value = in;
    else if (dec && !zero) next_value = value - 1'b1;
    else next_value = 4'b0; // default is clear
  end

assign zero = ~|value;

```

More Sophisticated Style

Combine input logic to register with register



```

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;
```

Summary

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?