The Verilog Hardware Description Language - A Dataflow View

Overview
In this lesson we will

✓ Introduce and explore the Verilog dataflow or Register Transfer Level (RTL) model.
✓ Introduce the dataflow operators.
✓ Study the continuous assignment dataflow model.
✓ Examine real-world effects under the dataflow model.
✓ Study Verilog dataflow level models of combinational and sequential circuits.

Introduction
As we know circuits and systems we are developing today
Growing in capability and complexity every day
Yesterday a sketch on a piece of paper and a handful of parts
Sufficient to try out a design idea
Today that is no longer possible
Idea is
• Modeled using computer based tools and languages
• Synthesized into the desired hardware implementation
• Test and verify the design
We use two key words here model and synthesize
While test is important
It applies no matter what approach is used

The Dataflow Model
Gate level modeling is an effective approach for working with smaller problems
Such an approach directly follows typical detailed logic diagram
Thus simplifies moving
From design
To model and simulation

Today embedded applications
Continually increasing in complexity
SSI and MSI modules of yesterday
    Being replaced by ASICs, FPGAs, and microprocessors
Developing a complete design at the gate level
    No longer feasible
Working at the gate level
    Similar writing sophisticated application in assembler
        Can be done but is impractical
That said – HDL synthesizers are not at same level of maturity as compilers
    • Compiler generally has one thing to worry about
    • Synthesizer has many – for example
        ✓ Critical path timing
        ✓ Race conditions
        ✓ Crosstalk
        ✓ Noise

On other hand powerful tools available or being developed
    To support design and modeling process
    Timing and power analyzers are examples

Developing at a higher level
    Not without problems
Farther that one moves
    From the low level details and
    Increases reliance on tools
        To produce those details
    The greater the risk that the tools
        Will produce less than optimum design
Result - designer must thoroughly understand
    Tools and their limitations
    Resulting design
Ability to push the limits of a design and a technology

   Comes from
       Years of experience
       Understanding of the problem
   Can’t find solution to all problems using Google search

Tools
   Can help us to solve the majority of the design problems
   Not sufficiently advanced to solve all autonomously

Dataflow Modeling
   Views a design from the perspective of
       Data moving through the system
           From source to destination
   In the digital world
       Such a view often referred to as RTL or register transfer level design
Contemporary tools able to accept a dataflow model as input
   Produce a low-level logic gate implementation
       Through a process called logic synthesis
Operators
   Syntax and operators used in Verilog at the dataflow level
       Follow that of the C language very closely
   Table below gives the most commonly used operators
Continuous Assignment
At the dataflow level
Design is modeled as movement of data
From module to module – functional block to functional block
To affect the application
That data moves over a net
Thus, a fundamental element of such modeling
Is ability to drive a value
From a source module
Onto the interconnecting net
To the destination modules

In Verilog such ability
Expressed by *continuous assignment*
This models continuous signal flow
Input changes → Output changes

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+</td>
<td>Add</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Subtract</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>Divide</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>Modulus</td>
</tr>
<tr>
<td>Relational</td>
<td>&gt;</td>
<td>Greater Than</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td>Greater Than or Equal</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>Less Than or Equal</td>
</tr>
<tr>
<td>Equality</td>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td></td>
<td>!=</td>
<td>Not Equal</td>
</tr>
<tr>
<td>Logical</td>
<td>!</td>
<td>Logical Negation</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitwise</td>
<td>~</td>
<td>Bitwise Negation</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift</td>
<td>&lt;&lt;</td>
<td>Shift Left</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;</td>
<td>Shift Right</td>
</tr>
</tbody>
</table>
Continuous assignment statement
Specified using the following syntax

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>assign destination net = source net expression</td>
</tr>
</tbody>
</table>

Left hand side of the continuous assignment
Must be
Scalar or vector (multiple lines) net
Right hand side of the expression
Can be a net, register, or function call return
Must be of the same size as the left hand side

Scalar cannot be assigned to a vector
Vice versa, for example.

- A continuous assignment is *always* active

Key point
Change on the right hand side forces evaluation of the left hand side
With the resulting assignment of the right hand side value
To the left hand side net

Combinational Logic
We illustrate a combinational dataflow model
Using the AndOr circuit designed earlier
That model using the continuous assignment
Expressed in the adjacent code fragment

Implementation using the bitwise AND and OR operators
Should be familiar from work with C counterparts

```vhdl
// continuous assignment
module AndOr(AandB, AorB, A, B);
output AandB, AorB;
input A, B;
wire AandB, AorB;
assign AandB = A&B;
assign AorB = A|B;
endmodule
```

The Real-World Affects – Part 2
Delays
Moving up one level of abstraction from the gate level
Does not preclude need to model real world effects on circuit behavior
Such effects remain important
The Verilog model for delay at the dataflow level
Follows naturally from that at the gate level
The syntax is given as

**Syntax**

```verilog
assign #delay net
```

Effect follows naturally – at the gate level

The model for the AndOr circuit designed earlier
Can include delays as seen in the following code fragment

```verilog
// continuous assignment
delay0 = 10;
wire AandB, AorB;
assign #delay0 AandB = A&B;
assign #delay0 AorB = A|B;
endmodule
```

The outputs of the system
Will now change 10 time units after either of the input signals changes

<table>
<thead>
<tr>
<th>Time,</th>
<th>A,</th>
<th>B,</th>
<th>AandB,</th>
<th>AorB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,</td>
<td>1,</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>0,</td>
<td>1,</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>0,</td>
<td>0,</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>0,</td>
<td>1,</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0,</td>
<td>1,</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Rise and fall time delays incorporated in a similar manner
Will discuss in greater detail later
Syntax for all three is given as

```
Syntax
assign #(rise time, fall time, delay) net
rise time: 0→1, fall time 1→0
in changing tristate device to / from tristate
delay is called turn-off/on delay
```

The model for the AndOr circuit designed earlier
Can include the delays

```
// Compute the logical AND and OR of inputs A and B.
module AndOr(AandB, AorB, A, B);
output AandB, AorB;
input A, B;
wire AandB, AorB;
parameter delay0 = 10;
parameter rise = 5;
parameter fall = 7;
assign #(rise, fall, delay0) AandB = A&B;
assign #(rise, fall, delay0) AorB = A|B;
endmodule
```

Outputs of the system
Now change 10 time units after either of the input signals changes
Reflect the rise and fall times
Shown as digital delay
Prop delay not shown in following output

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>AandB</th>
<th>AorB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Change in A
Reflected here
Sequential Logic

Following three code modules

Evolve the dataflow implementations of

Gated SR latch – gate level diagram and dataflow model illustrated

```
// Gate Level diagram
// Gated SR Latch with clear

module gsrLatch(q, qnot, sg, rg, clr, enab);
    input sg, rg, clr, enab;
    output q, qnot;
    wire rL, sL;
    assign rL = rg & clr & enab;
    assign sL = sg & clr & enab;

    // Build the basic RS latch
    assign q = ~(rL | ~clr | qnot);
    assign qnot = ~(sL | q);
endmodule
```

Master-slave SR flip-flop

```
// Dataflow Level Model
// Gated SR Latch
module gsrLatch(q, qnot, sg, rg, clr, enab);
    input sg, rg, clr, enab;
    output q, qnot;
    wire rL, sL;

    // Build the gating logic
    assign rL = rg & clr & enab;
    assign sL = sg & clr & enab;

    // Build the basic RS latch
    assign q = ~(rL | ~clr | qnot);
    assign qnot = ~(sL | q);
endmodule
```

```
// Use two SR Latches in a master slave configuration to build a flip-flop
module srmsff(q, qnot, s, r, clk, clr);
    input s, r, clk, clr;
    output q, qnot;
    gsrLatch master(qm, qmnot, s, r, clr, clk);
    gsrLatch slave(q, qnot, qm, qmnot, clr, ~clk);
endmodule
```
Two bit binary counter designed earlier

// Build a two bit binary up counter using master slave SR flip-flops

module TwoBitCntr(qA, qB, clr, clk);
input clr, clk;
output qA, qB;
wire sA, rA;
wire qA, qAnot, qB;
assign sA = qAnot & qB;
assign rA = qA & qB;
srmsff FFB(qB, qBnot, qBnot, qB, clk, clr);
srmsff FFA(qA, qAnot, sA, rA, clk, clr);
endmodule

Summary
In this lesson we
  ✔ Introduced and explored the Verilog dataflow or Register Transfer Level (RTL) model.
  ✔ Examined the dataflow operators.
  ✔ Worked with the continuous assignment dataflow model.
  ✔ Examined real-world effects under the dataflow model.
  ✔ Developed Verilog dataflow level models of combinational and sequential circuits.