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

Operator	*Symbol*	*Operation*
Arithmetic	+	Add
	-	Subtract
	/	Divide
	*	Multiply
	%	Modulus
Relational	>	Greater Than
	<	Less Than
	>=	Greater Than or Equal
	<=	Less Than or Equal
Equality	==	Equal
	!=	Not Equal
Logical	!	Logical Negation
	&&	Logical AND
	||	Logical OR
Bitwise	~	Bitwise Negation
	&	Bitwise AND
	|	Bitwise OR
Shift	<<	Shift Left
	>>	Shift Right

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

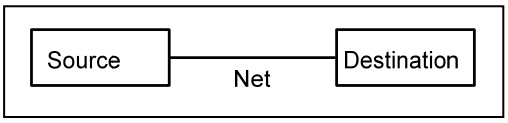
Thus, a fundamental element of such modeling

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

Continuous assignment statement

Specified using the following syntax

Syntax

assign destination net = source net expression

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

```
//  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
```

Implementation using the bitwise AND and OR operators

Should be familiar from work with C counterparts

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

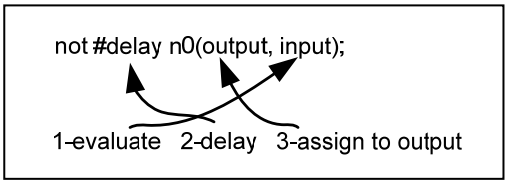
Follows naturally from that at the gate level

The syntax is given as

Syntax

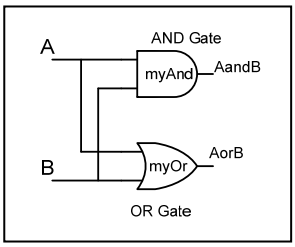
assign #delay net

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




```
// continuous assignment
module AndOr(AandB, AorB, A, B);
   output AandB, AorB;
   input A, B;

   wire AandB, AorB;
   parameter delay0 = 10;

   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

Time,	A,	B,	AandB,	AorB
0	1,	1,	x,	x
10	0,	1,	1,	1
20	0,	0,	0,	1
30	0,	1,	0,	0
40	0,	1,	0,	1

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;
   parmeter 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

	Time	A	B	AandB	AorB
	0	1	1	x	x
rise for A&B	5	1	1	1	1
	10	0	1	1	1
fall for A&B	17	0	1	0	1
	20	0	0	0	1
	27	0	0	0	0
	30	0	1	0	0
	35	0	1	0	1

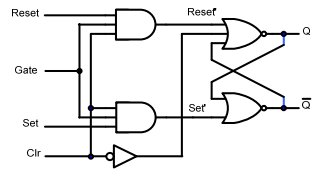
Change in A Reflected here

Sequential Logic

Following three code modules

Evolve the dataflow implementations of

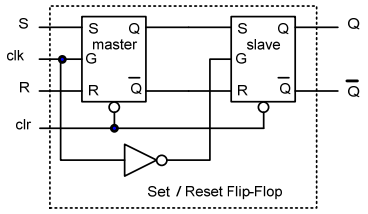
Gated SR latch – gate level diagram and dataflow model illustrated




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

```
// 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;
   wire q, qnot;

   // 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
```

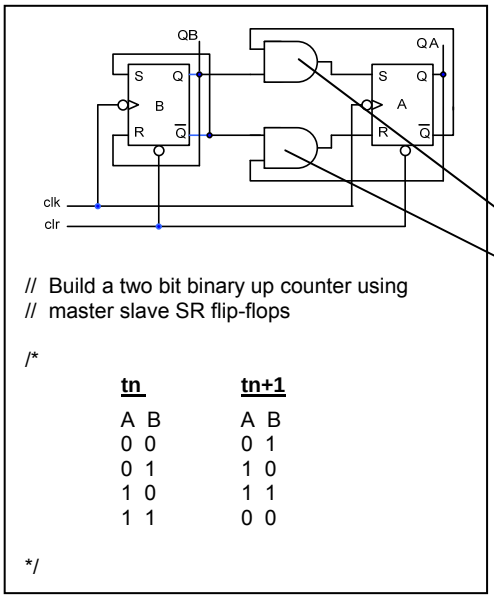
Master-slave SR flip-flop




```
// Use two SR Latches in a master slave
// configuration to build a flip-flop
```

```
// 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

/*
          tn          tn+1
          A  B        A  B
          0  0        0  1
          0  1        1  0
          1  0        1  1
          1  1        0  0

*/
```

```
// Build a synchronous 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.