Introduction

Hsi-Pin Ma

http://lms.nthu.edu.tw/course/38127
Department of Electrical Engineering
National Tsing Hua University

Outline

- Introduction
- Sample Design
- Structural Modeling
- RTL Modeling
- Logic Modeling and Simulation Using Xilinx ISE
- A Simple Example

Introduction

Hardware Description Language

- A high-level programming language offering special constructs to model microelectronic circuits
  - Describe the operation of a circuit at various level of abstraction
    - Behavior
    - Function
    - Structure
  - Describe the timing of a circuit
  - Express the concurrency of circuit operation

Levels of Abstraction (1/2)

Levels of Abstraction (2/2)

- Behavioral Level (Architectural/Algorithmic Level)
  - Describes a system by the flow of data between its functional blocks
  - Defines signal values when they change
- Register Transfer Level (Dataflow Level)
  - Describe a system by the flow of data and control signals between and within its functional blocks
  - Defines signal values with respect to a clock
  - RTL (Register Transfer Level) is frequently used for the Verilog description with the combination of behavioral and dataflow constructs which is acceptable to logic synthesis tools.
- Gate Level (Structural)
  - A model that describes the gates and the interconnections between them
- Transistor/Switch/Physical Level
  - A model that describes the transistors and the interconnections between them

VLSI Design Flow

Event Simulation of a Verilog Model

- Compilation
  - Compilation and elaboration
- Initialization
  - Initialize module parameters
    - Set other storage element to unknown (X) state
      - Unknown or un-initialized
    - Set undriven nets to the high-impedance (Z) state
      - Tri-state or floating
- Simulation

Verilog Simulation

Sample Design

Scenario

Device under Test (DUT)
Stimulus & Control
Response Generation & Verification
U0
A
B
a
b
out
sel
OUT
SEL
U1
A'
B'
a
b
out
sel
OUT'
SEL'
a
b
out
sel
SMUX
Module + Testbench

Logic Function in Verilog

- Combinational logic
  - logic function: $\text{out} = a \cdot \text{sel} + b \cdot \text{sel}'$
  - assign out = (a&sel) | (b&(~sel)) ;
- Basic logic operator
  - AND: &
  - OR: |
  - NOT: ~

Verilog Module

module name declaration

I/O definition

module functionality

```
module smux(out, a, b, sel);
output out;
input a,b,sel;

assign out = (a&sel) | (b&(~sel));

endmodule
```

Verilog Testbench

module name declaration

I/O definition

module instantiation

apply stimulus

```
module test_smux;

wire OUT;
reg A,B,SEL;

smux U0(.out(OUT),.a(A),.b(B),.sel(SEL));

initial
begin
  A=0;B=0;SEL=0;
  #10 A=0;B=0;SEL=1;
  #10 A=0;B=1;SEL=0;
  #10 A=0;B=1;SEL=1;
  #10 A=1;B=0;SEL=0;
  #10 A=1;B=0;SEL=1;
  #10 A=1;B=1;SEL=0;
  #10 A=1;B=1;SEL=1;
end

endmodule
```

Structural Modeling

Verilog Primitives

- and : Logical AND
- or : Logical OR
- not : Inverter
- buf : Buffer
- xor : Logical exclusive OR
- nand : Logical AND inverted
- nor : Logical OR inverted
- xnor : Logical exclusive OR inverted

Structural Modeling

```
module SMUX(out, a, b, sel);

output out;
input a,b,sel;
wire sel_n,t1,t2;

not U0(sel_n,sel);
and U1(t1,a,sel);
and U2(t2,b,sel_n);
or  U3(out,t1,t2);

endmodule
```

RTL Modeling

Operators (1/3)

Bitwise Operators		
OP	**Usage**	**Description**
~	~m	Invert each bit of m
&	m & n	AND each bit of m with each bit of n
|	m | n	OR each bit of m with each bit of n
^	m ^ n	Exclusive OR each bit of m with n
~^ or ^~	m ~^ n or m ^~ n	Exclusive NOR each bit of m with n

Unary Reduction Operators		
OP	**Usage**	**Description**
&	&m	AND all bits in m together (1-bit result)
~&	~&m	NAND all bits in m together (1-bit result)
|	|m	OR all bits in m together (1-bit result)
~|	~|m	NOR all bits in m together (1-bit result)
^	^m	Exclusive OR all bits in m (1-bit result)
~^ or ^~	~^m or ^~m	Exclusive NOR all bits in m (1-bit result)

Operators (2/3)

Arithmetic Operators		
OP	**Usage**	**Description**
+	m + n	Add n to m
-	m - n	Subtract n from m
-	-m	Negate m (2's complement)
*	m * n	Multiply m by n
/	m / n	Divide m by n
%	m % n	Modulus of m / n

The divisor for divide operator may be restricted to constants and a power of 2

Synthesis not supported

Logical Operators		
OP	**Usage**	**Description**
!	!m	Is m not true? (1-bit True/False result)
&&	m && n	Are both m and n true? (1-bit True/False result)
||	m || n	Are either m or n true? (1-bit True/False result)

Equality Operators (compares logic values of 0 and 1)		
OP	**Usage**	**Description**
==	m == n	Is m equal to n? (1-bit True/False result)
!=	m != n	Is m not equal to n? (1-bit True/False result)

Identity Operators (compares logic values of 0, 1, x, and z)		
OP	**Usage**	**Description**
===	m === n	Is m identical to n? (1-bit True/False result)
!==	m !== n	Is m not identical to n? (1-bit True/False result)

Synthesis not supported

Synthesis not supported

Operators (3/3)

Relational Operators		
OP	**Usage**	**Description**
<	m < n	Is m less than n? (1-bit True/False result)
>	m > n	Is m greater than n? (1-bit True/False result)
<=	m <= n	Is m less than or equal to n? (True/False result)
>=	m >= n	Is m greater than or equal to n? (True/False result)

Logical Shift Operators		
OP	**Usage**	**Description**
<<	m << n	Shift m left n-times
>>	m >> n	Shift m right n-times

Misc Operators		
OP	**Usage**	**Description**
? :	sel?m:n	If sel is true, select m: else select n
{}	{m,n}	Concatenate m to n, creating larger vector
{{}}	{n{m}}	Replicate m n-times

assign

- **assign** continuous construct
  - combinational logics

```
module SMUX (out,a,b,sel);
output out;
input a,b,sel;

  assign out = (a&sel) | (b&(~sel));

endmodule
```

This out has to be declared as “wire” or or “output” data type.
This expression can not be inside always @().

always

- **always** statements

```
module SMUX (out,s,b,sel);
output out;
input a,b,sel;
reg out;

always @*
   out = (a&sel) | (b&(~sel));

endmodule
```

This out has to be declared as “reg” data type.

Logic Modeling and Simulation Using Xilinx Vivado

Design Flow

- General design flow
  - Design construction
  - Behavioral simulation
  - Design implementation
  - Timing simulation
- HDL-based design Flow

Important Notes

- **Draw schematic first** and then construct Verilog codes.
- Verilog RTL coding philosophy is not the same as C programming
  - Every Verilog RTL construct has its own logic mapping (for synthesis)
  - You should have the logics (draw schematic) first and then the RTL codes
  - You have to write **synthesizable** RTL codes

Open Vivado

Vivado 2016.2
Vivado 2016.2
File Flow Tools Window Help
Search commands
VIVADO
HLx Editions
XILINX
ALL PROGRAMMABLE.
Quick Start
Create New Project
Open Project
Open Example Project
Tasks
Manage IP
Open Hardware Manager
Xilinx Tcl Store
Information Center
Documentation and Tutorials
Quick Take Videos
Release Notes Guide
Recent Projects
project_1
C:/Users/hsipin/project_1
Tcl Console

Open New Project (1/3)

Vivado 2016.2

File Flow Tools Window Help

VIVADO HLx Editions

Double Click

Quick Start

Create New Project | Open Project | Open Example Project

Tasks

Manage IP | Open Hardware Manager | Xilinx Tcl Store

Information Center

Documentation and Tutorials | Quick Take Videos | Release Notes Guide

Tcl Console

New Project

Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a

New Project

Project Name

Enter a name for your project and specify a directory where the project data

Project name: smux

Project location: C:/Users/hsipin

☑ Create project subdirectory

Project will be created at: C:/Users/hsipin/smux

Open New Project (2/3)

New Project

Project Type

Specify the type of project to create.

- RTL Project
  You will be able to add sources, create block designs in IP Integ
  and analysis.
  - Do not specify sources at this time
- Post-synthesis Project: You will be able to add sources, view de
  - Do not specify sources at this time
- I/O Planning Project
  Do not specify design sources. You will be able to view part/pac
- Imported Project
  Create a Vivado project from a Synplify, XST or ISE Project File.
- Example Project
  Create a new Vivado project from a predefined template.

New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts Boards

Filter

Product category: All

Speed grade: -1

Family: Artix-7

Temp grade: All Remaining

Package: cpg236

Reset All Filters

Search: xc7a35tcp (1 match)

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs	LUT Elements
xc7a35tcpg236-1	236	50	90	41600	2	2	106	20800

< Back | Next > | Finish | Cancel

New Project

VIVADO HLx Editions

New Project Summary

A new RTL project named 'lab1' will be created.

The default part and product family for the new project:
Default Part: xc7a35tcpg236-1
Product: Artix-7
Family: Artix-7
Package: cpg236
Speed Grade: -1

To create the project, click Finish

XILINX ALL PROGRAMMABLE.

? | < Back | Next > | Finish | Cancel

Finish

Open New Project (3/3)

smux - [C:/Users/hsipin/smux/smux.xpr] - Vivado 2016.2

File Edit Flow Tools Window Layout View Help

Search commands

Default Layout

Ready

Flow Navigator

- **Project Manager**
  - Project Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP Integrator
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- Simulation
  - Simulation Settings
  - Run Simulation
- RTL Analysis
  - Elaboration Settings
  - Open Elaborated Design
- Synthesis
  - Synthesis Settings
  - Run Synthesis
  - Open Synthesized Desig
- Implementation
  - Implementation Settings
  - Run Implementation
  - Open Implemented Desig

Project Manager - smux

Sources

- Design Sources
- Constraints
- Simulation Sources
  - sim_1

Hierarchy | Libraries | Compile Order

Properties

Select an object to see properties

Project Summary

Project Settings Edit

Project name:	smux
Project location:	C:/Users/hsipin/smux
Product family:	Kintex-7
Project part:	xc7k70tfbv676-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7k70tfbv676-1
Strategy:	Vivado Synthesis Defaults

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7k70tfbv676-1
Strategy:	Vivado Implementation Defaults

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1	constrs_1	Not started													
impl_1	constrs_1	Not started													

Tcl Console | Messages | Log | Reports | Design Runs

New Source (1/6)

New Source (2/6)

New Source (3/6)

New Source (4/6)

New Source (5/6)

smux - [C:/Users/hsipin/smux/smux.xpr] - Vivado 2016.2

File Edit Flow Tools Window Layout View Help

Flow Navigator

- Project Manager
  - Project Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP Integrator
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- Simulation
  - Simulation Settings
- RTL Analys
- Synthesis
- Implementa

Project Manager - smux

Sources

- Design Sources (1)
  - smux (smux.v)
- Constraints
- Simulation Sources (1)

Hierarchy | Libraries | Compile Order

Source File Properties

smux.v

Enabled

press and double click

Project Summary | smux.v

C:/Users/hsipin/smux/smux.srcs/sources_1/new/smux.v

```
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  ////////////////////////////////////////////////////////////////////////////////
21
22
23  module smux(
24      input a,
25      input b,
26      input c,
27      output out
28      );
29  endmodule
```

Project Summary | smux.v *

C:/Users/hsipin/smux/smux.srcs/sources_1/new/smux.v

```
18  // Additional Comments:
19  //
20  ////////////////////////////////////////////////////////////////////////////////
21
22
23  module smux(
24      input a,
25      input b,
26      input c,
27      output out
28      );
29
30  assign out = (a&sel) | (b&(~sel));
31
32  endmodule
33
```

THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed

23:0 | Insert | Verilog

New Source (6/6)

smux - [C:/Users/hsipin/smux/smux.xpr] - Vivado 2016.2

File Edit Flow Tools Window Layout View Help

- New Project...
- Open Project...
- Open Recent Project
- Open Example Project...
- Save Project As...
- Write Project Tcl...
- Archive Project...
- Close Project
- Open Checkpoint...
- Open Recent Checkpoint
- Write Checkpoint...
- New IP Location...
- Open IP Location...
- Open Recent IP Location
- New File...
- Open File... Ctrl+O
- Open Recent File
- Open IP-XACT File...
- Save File Ctrl+S
- Save File As...
- Save All Files...
- Add Sources... Alt+A
- Open Source File... Ctrl+N
- Import
- Export
- Launch SDK
- Open Log File
- Open Journal File
- Print... Ctrl+P
- Exit

Project Manager - smux

Design Sources (1)
smux (smux.v)
Constraints
Simulation Sources (1)
sim_1 (1)

Hierarchy Libraries Compile Order

Source File Properties
smux.v
Enabled
Location: C:/Users/hsipin/smux/smux.srcs/sources_1/new
General Properties

Project Summary | smux.v *

C:/Users/hsipin/smux/smux.srcs/sources_1/new/smux.v

```
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module smux(
    input a,
    input b,
    input c,
    output out
    );

assign out = (a&sel) | (b&(~sel));

endmodule
```

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1	constrs_1	Not started													
impl_1	constrs_1	Not started													

Tcl Console | Messages | Log | Reports | Design Runs

Remember to save files

Add Testbench (1/5)

Add Testbench (2/5)

Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Create Source File

Create a new source file and add it to your project.

File type: Verilog

File name: test_smux.v

File location: <Local to Project>

? OK Cancel

Add Files Add Directories Create File

Scan and add RTL include files into project

Copy sources into project

Add sources from subdirectories

Include all design sources for simulation

? < Back Next > Finish Cancel

Add Testbench (3/5)

Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Index	Name	Library	Location
1	test_smux.v	xil_defaultlib	<Local to Project>

Add Files | Add Directories | Create File

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [x] Add sources from subdirectories
- [x] Include all design sources for simulation

? | < Back | Next > | Finish | Cancel

Add Testbench (4/5)

Add Testbench (5/5)

smux - [C:/Users/hsipin/smux/smux.xpr] - Vivado 2016.2

File Edit Flow Tools Window Layout View Help

Sources

- Design Sources (1)
  - smux (smux.v)
- Constraints
- Simulation Sources (1)
  - sim_1 (1)
    - test_smux (test_smux.v) (1)
      - U0 - smux (smux.v)

Hierarchy | Libraries | Compile Order

C:/Users/hsipin/smux/smux.srcs/sim_1/new/test_smux.v

```
1  module test_smux;
2  wire OUT;
3  reg A,B,SEL;
4
5  smux U0(.out(OUT),.a(A),.b(B),.sel(SEL));
6
7  initial
8  begin
9    A=0;B=0;SEL=0;
10   #10 A=0;B=0;SEL=1;
11   #10 A=0;B=1;SEL=0;
12   #10 A=0;B=1;SEL=1;
13   #10 A=1;B=0;SEL=0;
14   #10 A=1;B=0;SEL=1;
15   #10 A=1;B=1;SEL=0;
16   #10 A=1;B=1;SEL=1;
```

Put your testbench here

Simulation (1/4)

smux - [C:/Users/hsipin/LD/smux/smux.xpr] - Vivado 2016.2

File Edit Flow Tools Window Layout View Run Help

Flow Navigator

- Project Manager
  - Project Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP Integrator
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- Simulation
  - Simulation Settings
  - Run Simulation
- RTL Analysis
  - Elaboration S
  - Open Elabora
- Synthesis
  - Synthesis Settings
  - Run Synthesis
  - Open Synthesized Desig
- Implementation
  - Implementation Settings
  - Run Implementation
  - Open Implemented Desig

Press and Select

Run Behavioral Simulation
Run Post-Synthesis Functional Simulation
Run Post-Synthesis Timing Simulation
Run Post-Implementation Functional Simulation
Run Post-Implementation Timing Simulation

Behavioral Simulation - Functional - sim_1 - test_smux

Sources

- Design Sources (1)
  - smux (smux.v)
- Constraints
- Simulation Sources (1)
  - sim_1 (1)
    - test_smux (test_smux.v) (1)
      - U0 - smux (smux.v)

Objects

Name	Value	Data Typ
OUT	1	Logic
A	1	Logic
B	1	Logic
SEL	1	Logic

smux.v | test_smux.v

C:/Users/hsipin/LD/smux/smux.srcs/sim_1/new/test_smux.v

```
1  module test_smux;
2  wire OUT;
3  reg A,B,SEL;
4
5  smux U0(.out(OUT),.a(A),.b(B),.sel(SEL));
6
7  initial
8  begin
9    A=0;B=0;SEL=0;
10   #10 A=0;B=0;SEL=1;
11   #10 A=0;B=1;SEL=0;
12   #10 A=0;B=1;SEL=1;
13   #10 A=1;B=0;SEL=0;
14   #10 A=1;B=0;SEL=1;
15   #10 A=1;B=1;SEL=0;
16   #10 A=1;B=1;SEL=1;
17 end
18
19 endmodule
```

```
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'test_smux_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:12 . Memory (MB): peak = 780.797 ; gain = 10.039
```

Type a Tcl command here

Tcl Console | Messages | Log

Vivado Simulator

Simulation (2/4)

smux - [C:/Users/hsipin/LD/smux/smux.xpr] - Vivado 2016.2

File Edit Flow Tools Window Layout View Run Help

Press maximize

Flow Navigator

- Project Manager
  - Project Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP Integrator
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- Simulation
  - Simulation Settings
  - Run Simulation
- RTL Analysis
  - Elaboration Settings
  - Open Elaborated Design
- Synthesis
  - Synthesis Settings
  - Run Synthesis
  - Open Synthesized Design
- Implementation
  - Implementation Settings
  - Run Implementation
  - Open Implemented Design

Behavioral Simulation - Functional - sim_1 - test_smux

Scopes

Name	Design Unit	Block Type
test_smux	test_smux	Verilog Module
U0	smux	Verilog Module
glbl	glbl	Verilog Module

Objects

Name	Value	Data Typ
OUT	1	Logic
A	1	Logic
B	1	Logic
SEL	1	Logic

smux.v | test_smux.v | Untitled 2

Name	Value
OUT	1
A	1
B	1
SEL	1

/test_smux/SEL

Tcl Console

```
# }
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'test_smux_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 799.906 ; gain = 0.000
```

Type a Tcl command here

Tcl Console | Messages | Log

Sim Time: 1 us

Simulation (3/4)

Behavioral Simulation - Functional - sim_1 - test_smux

smux.v | test_smux.v | Untitled 2

Scope | Objects

Name	Value
OUT	1
A	1
B	1
SEL	1

1,000,000 ps

1,000,000 ps | 1,000,001 ps | 1,000,002 ps | 1,000,003 ps

Use these to zoom in or zoom out

Tcl Console

Scroll this to the beginning

Simulation (4/4)

smux - [C:/Users/hsipin/LD/smux/smux.xpr] - Vivado 2016.2

File Edit Flow Tools Window Layout View Run Help

Flow Navigator

- Project Manager
  - Project Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP Integrator
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- Simulation
  - Simulation Settings
  - Run Simulation
- RTL Analysis
  - Elaboration Settings
  - Open Elaborated Design
- Synthesis
  - Synthesis Settings
  - Run Synthesis
  - Open Synthesized Design
- Implementation
  - Implementation Settings
  - Run Implementation
  - Open Implemented Design

Behavioral Simulation - Functional - sim_1 - test_smux

smux.v | test_smux.v | Untitled 2

Close Simulation

Name	Value
OUT	0
A	0
B	0
SEL	0

0.000 ns

0 ns, 20 ns, 40 ns, 60 ns, 80 ns, 100 ns, 120 ns

Tcl Console

Sim Time: 1 us

A Combinational Logic Example

Design Procedure

1. From the ***specifications***, determine the inputs, outputs, and their symbols.
2. Derive the ***truth table*** (***functions***) from the relationship between the inputs and outputs
3. Derive the ***simplified Boolean functions*** for each output function.
4. Draw the logic diagram.
5. Construct the Verilog code according to the logic diagram.
6. Write the testbench and verify the design.

$$F(x,y,z) = \sum(1,4,5,6,7) = f$$

1 input: x,y,z output: f

2

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

3

f=F(x,y,z)=x+y′z

4

$$F(x,y,z) = \sum(1,4,5,6,7) = f$$

4

f=F(x,y,z)=x+y'z

5

```
module ex(f,x,y,z);
output f;
input x,y,z;

  assign f = x | ((~y)&z);

endmodule
```

6

```
module t_ex;
wire f1;
reg x1,y1,z1;

ex U0(.f(f1),.x(x1),.y(y1),.z(z1));

initial
begin
  x1=0;y1=0;z1=0;
  #5 x1=0;y1=0;z1=1;
  #5 x1=0;y1=1;z1=0;
  #5 x1=0;y1=1;z1=1;
  #5 x1=1;y1=0;z1=0;
  #5 x1=1;y1=0;z1=1;
  #5 x1=1;y1=1;z1=0;
  #5 x1=1;y1=1;z1=1;
  #5 x1=0;y1=0;z1=0;
end

endmodule
```

$$F(x, y, z) = \sum(1, 4, 5, 6, 7) = f$$

Example 2 (1/3)

$$f_1 = x'y'z + xy'z' + xyz$$
$$= m_1 + m_4 + m_7 = \sum(1, 4, 7)$$

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Module definition

```
module exp1(f1, x, y, z);
output f1;
input x,y,z;

assign f1 = ((~x)&(~y)&z) | (x&(~y)&(~z) | x&y&z);

endmodule
```

Example 2 (2/3)

- Testbench

```
module test_f1;
wire out;
reg a,b,c;

exp1 U0(.f1(out),.x(a),.y(b),.z(c));

initial
begin
  a=0;b=0;c=0;
  #10 a=0;b=0;c=1;
  #10 a=0;b=1;c=0;
  #10 a=0;b=1;c=1;
  #10 a=1;b=0;c=0;
  #10 a=1;b=0;c=1;
  #10 a=1;b=1;c=0;
  #10 a=1;b=1;c=1;
end

endmodule
```

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Example 2 (3/3)

- Simulation Results

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Example 3 (1/3)

$$
\begin{aligned}
f_2 &= x'yz + xy'z + xyz' + xyz \\
&= m_3 + m_5 + m_6 + m_7 = \sum(3,5,6,7)
\end{aligned}
$$

x	y	z	f_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Module definition

```
module exp2(f2, x, y, z);
output f2;
input x,y,z;

assign f2 = ((~x)&y&z) | (x&(~y)&z) |( x&y&(~z))| (x&y&z);

endmodule
```

Example 3 (2/3)

- Testbench

```
module test_f2;
wire out;
reg a,b,c;

exp2 U0(.f2(out),.x(a),.y(b),.z(c));

initial
begin
  a=0;b=0;c=0;
  #10 a=0;b=0;c=1;
  #10 a=0;b=1;c=0;
  #10 a=0;b=1;c=1;
  #10 a=1;b=0;c=0;
  #10 a=1;b=0;c=1;
  #10 a=1;b=1;c=0;
  #10 a=1;b=1;c=1;
end

endmodule
```

x	y	z	f_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Example 3 (3/3)

- Simulation Results

x	y	z	f_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Verilog Module Construction (1/2)

- Separate flip-flops with other logics (two types)
  - flip-flops (edge-triggered with clock, reset)
  - combinational logics (level sensitive)
- Combinational logics
  - simple logics (AND, OR, NOT)
  - coder/decoder (mapping, addressing)
  - comparison (conditional/equality test)
  - selection (select correct results, MUX)
  - arithmetic functions and superposition (+,-,*,binary shift)
- Finite state machine (FSM)

Verilog Module Construction (2/2)

- Separate flip-flops with other logics
  - For a D-type flip-flop

```
always @(posedge clk or negedge rst_n)
  if (~rst_n)
    q<=0;
  else
    q<=d;
```

  - For a 2-to-1 MUX

```
always @*
  if (select==1'b1)
    out=a;
  else
    out=b;
```

```
assign out = (select==1'b1) ? a : b;
```

A 2-to-4 Line Decoder

a_1	a_0	b_3	b_2	b_1	b_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

```
module Dec24(a, b);
input [1:0] a; //binary input (2 bits wide)
output [3:0] b;  // one-hot output (4 bits wide)

assign b[3]=a[1]&a[0];
assign b[2]=a[1]&(~a[0]);
assign b[1]=(~a[1])&a[0];
assign b[0]=(~a[1])&(~a[0]);

endmodule
```

$$b_3 = a_1 a_0$$
$$b_2 = a_1 a_0'$$
$$b_1 = a_1' a_0$$
$$b_0 = a_1' a_0'$$

A BCD to Seven-Segment Display Decoder

```
// define segment codes
`define SS_0 7'b1111110
`define SS_1 7'b0110000
`define SS_2 7'b1101101
`define SS_3 7'b1111001
`define SS_4 7'b0110011
`define SS_5 7'b1011011
`define SS_6 7'b1011111
`define SS_7 7'b1110000
`define SS_8 7'b1111111
`define SS_9 7'b1111011
```

```
module sseg(segs, bin);
output [6:0] segs;
input [3:0] bin;
reg [6:0] segs;

always @*
  case (bin)
    4'd0: segs = `SS_0;
    4'd1: segs = `SS_1;
    4'd2: segs = `SS_2;
    4'd3: segs = `SS_3;
    4'd4: segs = `SS_4;
    4'd5: segs = `SS_5;
    4'd6: segs = `SS_6;
    4'd7: segs = `SS_7;
    4'd8: segs = `SS_8;
    4'd9: segs = `SS_9;
    default: segs = 7'b0000000;
  endcase
endmodule
```

Maximun Unit

$$y = \max\{a, b\}$$

```
module Max(a, b, y);
input [3:0] a, b; //two input variables (4 bits wide)
output reg [3:0] y;  // maximum output (4 bits wide)

assign gt=(a>b) ? 1 : 0;
always @*
  if (gt)
    y = a;
  else
    y = b;

endmodule
```

```
always @*
  if (a > b)
    y = a;
  else
    y = b;
```

Ripple-Carry Adder (1/2)

```
module adder(a, b, cin, cout, s);
parameter n=4;
input [n-1:0] a, b;
input cin;
output reg [n-1:0] s;
output cout;
reg [n-1:0] c;
integer i;

assign cout = c[4];
always @*
  c[0] = cin;
always @*
  for (i=0;i<4;i=i+1)
    {c[i+1],s[i]} = a[i] + b[i] + c[i];

endmodule
```

```
assign {c[1],s[0]} = a[0] + b[0] + c[0];
assign {c[2],s[1]} = a[1] + b[1] + c[1];
assign {c[3],s[2]} = a[2] + b[2] + c[2];
assign {c[4],s[3]} = a[3] + b[3] + c[3];
```

Multi-bit Binary Adder (2/2)

behavioral model

```
module Adder1(a, b, cin, cout, s);
parameter n=8;
input [n-1:0] a, b;
input cin;
output [n-1:0] s;
output cout;

assign {cout, s} = a + b + cin;

endmodule
```