# (The Course) Verilog Environment

Carmi Merimovich

The Academic College of Tel-Aviv

January 29, 2025

# Verilog

- Circuits drawings are not useful as input to programs
- We have hardware description languages (HDLs)
  - Two languages are mainstream
  - The one we will **not** use is `VHDL`
  - It is `ADA` like, I hate it
- `Verilog` is the language we will use
- `Verilog` is **not** a programming language
- The Web is choke full of material on Verilog
- Books titled 'digital design' and 'Verilog' will also do
- This chapter aims to show how to use the software
- Definitely **not** to teach Verilog

# Software Needed

- `iverilog` - Verilog compiler
- `vvp` - Circuit emulator
- `GTKWave` - Waveform viewer
- gmake/make - (recommended) make

## Installing

- All of the programs above are GPL'ed
- There is an easy Windows installer
- The programs are in the `bin` folders
- You can add them to your `path` (I did not)
- `gmake` can be found in, e.g., strawberry Perl
  - folder: `C:\Strawberry\c\bin`

# Verilog Levels





## Levels used in this course

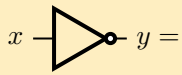
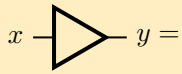
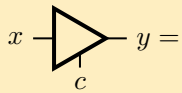
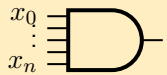
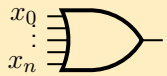
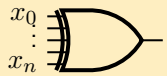
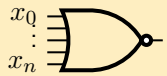
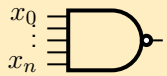
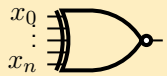
- Gate level - Will be used to synthesize circuits
- Data flow - Will be used for testbenches
- Behavioral - If there is no other way

## The reasons for the above are pedagogical

Namely, in a first course, students should use only gates

# Verilog Gate Primitves



| Gate | Verilog Primitive |
|---|---|
|   | `not(y,x)` |
|   | `buf(y,x)` |
|   | `bufif1(y,x,c)` |
|   | `and(y,x0,...,xn)` |
|   | `or(y,x0,...,xn)` |
|   | `xor(y,x0,...,xn)` |
|   | `nor(y,x0,...,xn)` |
|   | `nand(y,x0,...,xn)` |
|   | `xnor(y,x0,...,xn)` |

**Do not** use the primitives directly

There is no delay when using them

# Demo: a 3-input Gate with Delay

# Introducing the Following Features

- Gate primitve invocation
- Module definition

# Demo: and3

## Gate Level Example

Listing 1: demo/1/and3.v

```
‘timescale 1ns/1ns

module and3(
  output xyz,
  input  x,
  input  y,
  input  z);

  and #(7ns) (xyz,x,y,z);

endmodule
```

# Notes on the above

- Without the optional `timescale`, `iverilog` has issues
- As it stands we have defined a circuit
- No input, no output, no nothing
- We **must** check it
- We will add a **testbench**
- Testbneches are simulations, we use data flow for them

# Introducing the following Features

- Multiple modules in the same `.v` files
- Module instancing
- `logic` definition
- `wire` definition
- `assign` statement
- `repeat` statement
- `initial` block
- `$dumpfile`, `$dumpvars`, `$finish`

# Demo: and3 with testbench I



Listing 2: demo/2/and3.v

```
‘timescale 1ns/1ns

module and3(
  output xyz,
  input  x,
  input  y,
  input  z);

  and #(7ns) (xyz,x,y,z);

endmodule

module and3_tb;
```

# Demo: and3 with testbench II

```
logic x,y,z;  // Simulator thing
wire t;
and3 a(t,x,y,z);


initial begin
  $dumpfile("and3.vcd");
  $dumpvars;

  x = 0;
  y = 0;
  z = 0;

  repeat (2) begin
```

# Demo: and3 with testbench III

```
      repeat (2) begin
        repeat (2) begin
          #20ns;
          $display(x,y,z,t);
          z = z + 1;
        end
        y = y + 1;
      end
      x = x + 1;
    end
    $finish();
  end
endmodule
```

# Compiling



## Compiling `and3.v` $\Longrightarrow$ `and3`

```
c:\iverilog\bin\iverilog -g2012 -oand3 and3.v
```

- -gyear - Verilog standard year
- -o output - Default is `a.out`
- file(s) - list of `Verilog` files to compile

## Root module

- Modules not instanced will be root instanced
- `-sroot` can be used to specify the root module

# Simulating

## Simulate and3 $\Longrightarrow$ and3.vcd (due to $dumpfile)

```
c:\iverilog\bin\vvp and3
```

## View and3.vcd

```
c:\iverilog\gtkwave\bin\gtkwave and3
```

(Carmi) Lecture 7 reached here

Listing 3: demo/2/makefile

```
dir     = c:/iverilog/bin/
dir2    = c:/iverilog/gtkwave/bin/
module  = and3

all: $(module).vcd

$(module): $(module).v
  $(dir)iverilog \
    -g2012 \
    -o $(module) \
    $(module).v

$(module).vcd: $(module)
```

# makefile II

```
  $(dir)vvp $(module)
  $(dir2)gtkwave $(module).vcd

clean:
  -del  $(module)
  -del  $(module).vcd
```

# Splitting to Several Files: and3



Listing 4: demo/3/and3.v

```
`timescale 1ns/1ns

module and3(
  output xyz,
  input  x,
  input  y,
  input  z);

  and #(7ns) (xyz,x,y,z);

endmodule
```

# Splitting to Several Files: and3_tb I





Listing 5: demo/3/and3_tb.v

```
`timescale 1ns/1ns

module and3_tb;

  logic x,y,z;
  wire t;
  and3 a(t,x,y,z);


  initial begin
    $dumpfile("and3.vcd");
    $dumpvars;
```

# Splitting to Several Files: and3_tb II

```
    x = 0;
    y = 0;
    z = 0;

    repeat (2) begin
      repeat (2) begin
        repeat (2) begin
          #20ns;
          $display(x,y,z,t);
          z = z + 1;
        end
        y = y + 1;
      end
      x = x + 1;
    end
```

# Splitting to Several Files: and3_tb III

```
    $finish();
  end
endmodule
```

# Compiling Multifile Project





## Two options

- List of files:

```
c:\iverilog\bin\iverilog -g2012 -oand3 and3.v and3_tb.v
```

- Command file

```
c:\iverilog\bin\iverilog -g2012 -oand3 -cand3.cf
```

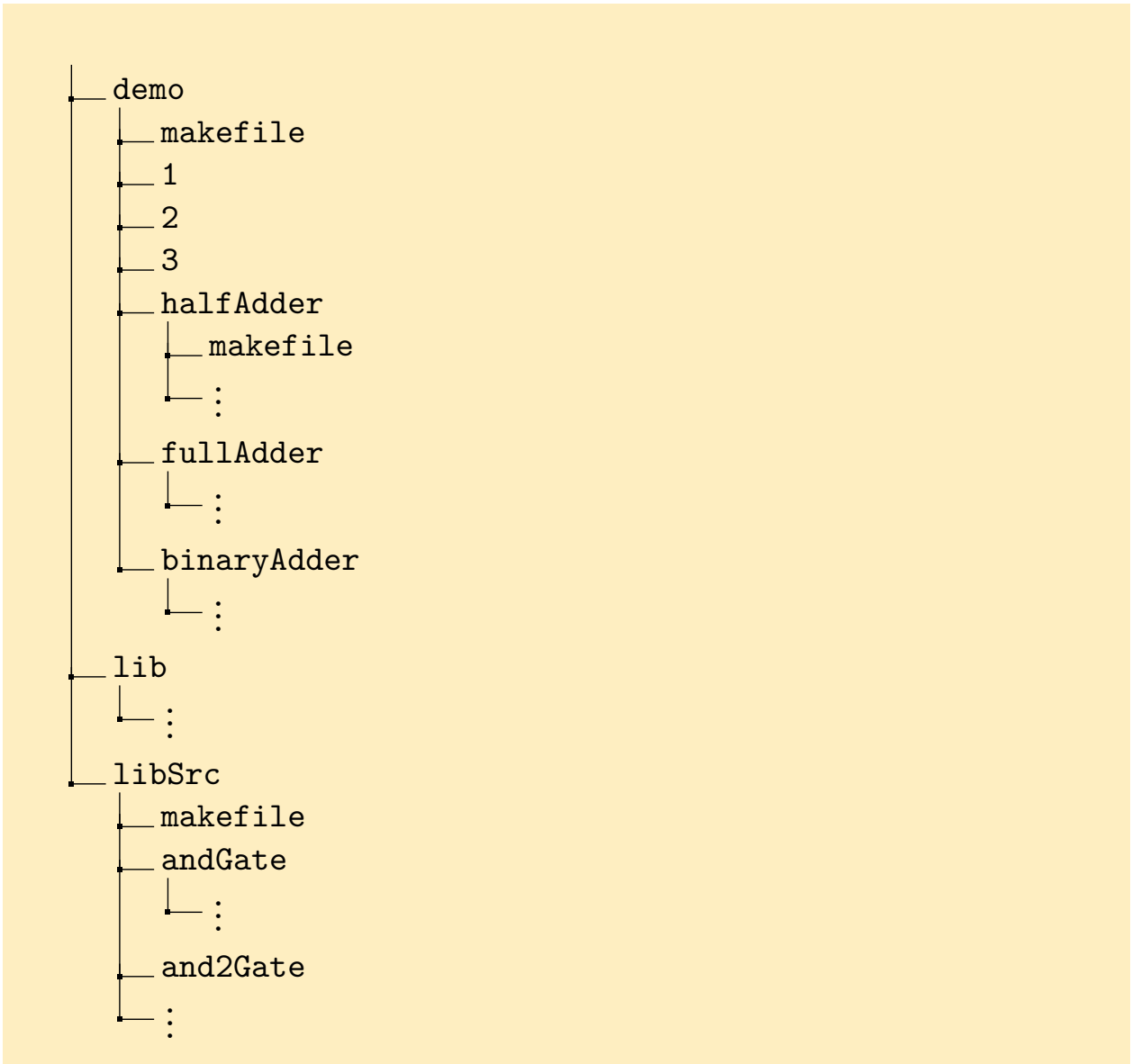
and in the file `and3.cf`

```
and3.v
and3_tb.v
```

## verilog.zip and demo.zip

(extraction of .zip files from the site)

# Folder Structure



# Library

- A folder can be designated a library with the `-y` option
- A file in the library should contain only **one** module
- The file name and the module name must be **identical**
- A library is searched when an instanced module is not found

## We supply a library

- It contains gates with delays
- It is in the folder `lib`
- The source code, so to speak, is in the `verilog` folder

# Delays

## Do not be misled

- The library delays are in order to get the gist of delays
- The delays are technology specific
- Usually the transitions times of $0 \rightarrow 1$ and $1 \rightarrow 0$ are different
- (`Verilog` has the machinery for different delays)

# Library Modules

| 2-pins | 3-pins | 4-pins | 5-pins |
|---|---|---|---|
| andGate | and3Gate | and4Gate | and5Gate |
| orGate | or3Gate | or4Gate | or5Gate |
| xorGate | xor3Gate | xor4Gate | xor5Gate |
| nandGate | nand3Gate | nand4Gate | nand5Gate |
| norGate | nor3Gate | nor4Gate | nor5Gate |

- 1-pin: notGate, bufGate, bufifGate
- $n$-pins: andMulti, orMulti, xorMulti, nandMulti, norMulti

# Completing the Motivation

The previous chapter circuits in `Verilog`

The code is in the `demo` folder

# Seven-Segment (formulae)

$$
\begin{aligned}
m_0 &= \bar{b}_3\bar{b}_2\bar{b}_1\bar{b}_0 \\
m_1 &= \bar{b}_3\bar{b}_2\bar{b}_1 b_0 \\
m_2 &= \bar{b}_3\bar{b}_2 b_1\bar{b}_0 \\
m_3 &= \bar{b}_3\bar{b}_2 b_1 b_0 \\
m_4 &= \bar{b}_3 b_2\bar{b}_1\bar{b}_0 \\
m_5 &= \bar{b}_3 b_2\bar{b}_1 b_0 \\
m_6 &= \bar{b}_3 b_2 b_1\bar{b}_0 \\
m_7 &= \bar{b}_3 b_2 b_1 b_0 \\
m_8 &= b_3\bar{b}_2\bar{b}_1\bar{b}_0 \\
m_9 &= b_3\bar{b}_2\bar{b}_1 b_0
\end{aligned}
$$

$$
\begin{aligned}
a &= \bar{m}_1\bar{m}_4\bar{m}_6 \\
b &= \bar{m}_5\bar{m}_6 \\
c &= \bar{m}_2 \\
d &= \bar{m}_1\bar{m}_4\bar{m}_7 \\
e &= \bar{m}_1\bar{m}_3\bar{m}_4\bar{m}_5\bar{m}_7\bar{m}_9 \\
f &= \bar{m}_1\bar{m}_2\bar{m}_3\bar{m}_7 \\
g &= \bar{m}_0\bar{m}_1\bar{m}_7
\end{aligned}
$$

# Seven Segment (module) I



Listing 6: segments7.v

```
‘timescale 1ns/1ns

module segments7(
  output a,
  output b,
  output c,
  output d,
  output e,
  output f,
  output g,
  input  n3,
  input  n2,
  input  n1,
```

# Seven Segment (module) II

```
  input  n0
);

  wire n0n, n1n, n2n, n3n;
  notGate ng0(n0n, n0);
  notGate ng1(n1n, n1);
  notGate ng2(n2n, n2);
  notGate ng3(n3n, n3);

  wire m0n,m1n,m2n,m3n,m4n,m5n,m6n,m7n;
  nand4Gate nnd0(m0n,n3n,n2n,n1n,n0n);
  nand4Gate nnd1(m1n,n3n,n2n,n1n,n0 );
  nand4Gate nnd2(m2n,n3n,n2n,n1 ,n0n);
  nand4Gate nnd3(m3n,n3n,n2n,n1 ,n0 );
  nand4Gate nnd4(m4n,n3n,n2 ,n1n,n0n);
```

# Seven Segment (module) III

```
nand4Gate nnd5(m5n,n3n,n2 ,n1n,n0 );
nand4Gate nnd6(m6n,n3n,n2 ,n1 ,n0n);
nand4Gate nnd7(m7n,n3n,n2 ,n1 ,n0 );
nand4Gate nnd8(m8n,n3 ,n2n,n1n,n0n);
nand4Gate nnd9(m9n,n3 ,n2n,n1n,n0 );

wire eTmp;
and3Gate anda(a,m1n,m4n,m6n);
andGate andb(b,m5n,m6n);
assign c = m2n;
and3Gate andd(d,m1n,m4n,m7n);
and4Gate ande1(eTmp,m1n,m3n,m4n,m5n);
and3Gate ande(e,eTmp,m7n,m9n);
and4Gate andf(f,m1n,m2n,m3n,m7n);
and3Gate andg(g,m0n,m1n,m7n);
```

# Seven Segment (module) IV

```
endmodule
```

# Seven Segment (testbench) I





Listing 7: segments7_tb.v

```
`timescale 1ns/1ns

module segments7_tb;
  logic x3,x2,x1,x0;
  wire a,b,c,d,e,f,g;

  segments7 m(a,b,c,d,e,f,g,x3,x2,x1,x0);

  initial begin
    $dumpfile("segments7.vcd");
    $dumpvars;

    x3 = 0;
```

# Seven Segment (testbench) II

```
    x2 = 0;
    x1 = 0;
    x0 = 0;

    repeat (2) begin
      repeat (2) begin
        repeat (2) begin
          repeat (2) begin
            #30ns;
            $display(x3,x2,x1,x0);
            carmi(a,b,c,d,e,f,g);
            $display();
            x0 = x0 + 1;
          end
          x1 = x1 + 1;
```

# Seven Segment (testbench) III

```
        end
        x2 = x2 + 1;
      end
      x3 = x3 + 1;
    end
    $finish();
  end

  task carmi(input a,b,c,d,e,f,g);
  if ((a == 1)  ||  (f == 1))
    $write("*");
  else
    $write(" ");

  if (a == 1)
```

# Seven Segment (testbench) IV

```
    $write("****");
  else
    $write("    ");

  if ((a == 1)  ||  (b == 1))
    $write("*\n");
  else
    $write(" \n");

  repeat (2) begin
    if (f == 1)
      $write("*    ");
    else
      $write("     ");
```

# Seven Segment (testbench) V

```
    if (b == 1)
      $write("*\n");
    else
      $write("\n");
  end

  if ((f == 1)  ||  (g == 1)  ||  (e == 1))
    $write("*");
  else
    $write(" ");

  if (g == 1)
    $write("****");
  else
    $write("    ");
```

# Seven Segment (testbench) VI

```
if ((b == 1) || (g == 1) || (c == 1))
  $write("*\n");
else
  $write(" \n");

repeat (2) begin
  if (e == 1)
    $write("*    ");
  else
    $write("     ");

  if (c == 1)
    $write("*\n");
  else
```

# Seven Segment (testbench) VII

```
      $write("\n");
  end


  if ((e == 1)  ||  (d == 1))
    $write("*");
  else
    $write(" ");

  if (d == 1)
    $write("****");
  else
    $write("    ");

  if ((d == 1)  ||  (c == 1))
```

# Seven Segment (testbench) VIII

```
    $write("*\n");
  else
    $write(" \n");

  endtask;

endmodule
```

# mod 3

(The circuits are from the logic gates chapter)

# mod 3 Functions

$$
\begin{aligned}
f_1 =& \bar{n}_3\bar{n}_2 n_1\bar{n}_0 + \bar{n}_3 n_2\bar{n}_1 n_0 + n_3\bar{n}_2\bar{n}_1\bar{n}_0 + n_3\bar{n}_2 n_1 n_0 + \\
& n_3 n_2 n_1\bar{n}_0 \\
f_0 =& \bar{n}_3\bar{n}_2\bar{n}_1 n_0 + \bar{n}_3 n_2\bar{n}_1\bar{n}_0 + \bar{n}_3 n_2 n_1 n_0 + n_3\bar{n}_2 n_1\bar{n}_0 + \\
& n_3 n_2\bar{n}_1 n_0
\end{aligned}
$$

# mod 3 (module) I



Listing 8: mod3.v

```
‘timescale 1ns/1ns

module mod3(
  output f1,
  output f0,
  input  n3,
  input  n2,
  input  n1,
  input  n0
);

  wire n3n, n2n, n1n, n0n;
  notGate not1(n3n, n3);
```

# mod 3 (module) II

```
notGate not2(n2n, n2);
notGate not3(n1n, n1);
notGate not4(n0n, n0);

wire m1, m2, m4, m5, m7, m8, m10,
     m11, m13, m14;

and4Gate and1(m1,   n3n,n2n,n1n,n0 );
and4Gate and2(m2,   n3n,n2n,n1 ,n0n);
and4Gate and4(m4,   n3n,n2 ,n1n,n0n);
and4Gate and5(m5,   n3n,n2 ,n1n,n0 );
and4Gate and7(m7,   n3n,n2 ,n1 ,n0);
and4Gate and8(m8,   n3 ,n2n,n1n,n0n);
and4Gate and10(m10, n3 ,n2n,n1 ,n0n);
and4Gate and11(m11, n3 ,n2n,n1 ,n0 );
```

# mod 3 (module) III

```
  and4Gate and13(m13, n3 ,n2 ,n1n,n0 );
  and4Gate and14(m14, n3 ,n2 ,n1 ,n0n);

  or5Gate o1(f1, m2, m5, m8, m11, m14);
  or5Gate o0(f0, m1, m4, m7, m10, m13);
endmodule
```

# mod 3 (testbench) I





Listing 9: mod3_tb.v

```
‘timescale 1ns/1ns

module mod3_tb;
  logic x3,x2,x1,x0;
  wire f1, f0;

  mod3 m(f1,f0,x3,x2,x1,x0);

  initial begin
    $dumpfile("mod3.vcd");
    $dumpvars;

    x3 = 0;
```

# mod 3 (testbench) II

```
    x2 = 0;
    x1 = 0;
    x0 = 0;

    repeat (2) begin
      repeat (2) begin
        repeat (2) begin
          repeat (2) begin
            #30ns;
            $display(x3,x2,x1,x0," ",f1,f0);
            x0 = x0 + 1;
          end
          x1 = x1 + 1;
        end
        x2 = x2 + 1;
```

# mod 3 (testbench) III

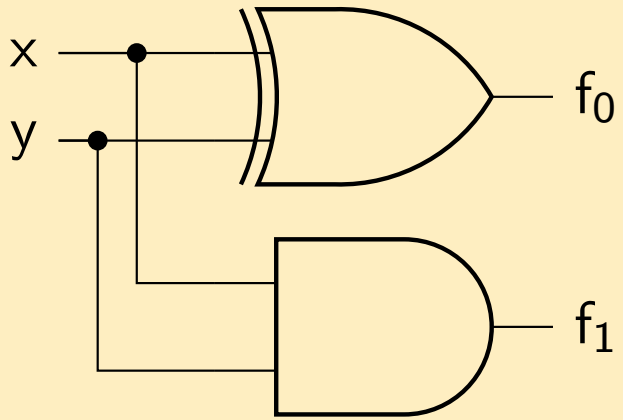
```
        end
        x3 = x3 + 1;
      end
      $finish();
    end

endmodule
```

(Carmi) Lecture 8 reached here

# Half Adder

The circuit is from the Logic Gates lecture

# Half Adder: Circuit



# Half Adder: Module

Listing 10: halfAdder.v

```
`timescale 1ns/1ns

module halfAdder(
    output f1,
    output f0,
    input  x,
    input  y
);
    andGate a1(f1,x,y);
    xorGate x1(f0,x,y);
endmodule
```

# Half Adder: Testbench I





Listing 11: halfAdder_tb.v

```
`timescale 1ns/1ns

module halfAdder_tb;
  logic x,y;
  wire c, s;

  halfAdder h(c, s, x, y);

  initial begin
    $dumpfile("halfAdder.vcd");
    $dumpvars;

    x = 0;
```

# Half Adder: Testbench II

```
    y = 0;
    repeat (2) begin
      repeat (2) begin
        #20ns;
        $display(x,y,c,s);
        y = y + 1;
      end
      x = x + 1;
    end
    $finish();
  end

endmodule
```

# 2-Bits Adder

# 2-Bits Binary Adder (Function)

$$
\begin{aligned}
f_2 =& \bar{x}_1 x_0 y_1 y_0 + x_1 \bar{x}_0 y_1 \bar{y}_0 + x_1 \bar{x}_0 y_1 y_0 + x_1 x_0 \bar{y}_1 y_0 + \\
& x_1 x_0 y_1 \bar{y}_0 + x_1 x_0 y_1 y_0 \\
f_1 =& \bar{x}_1 \bar{x}_0 y_1 \bar{y}_0 + \bar{x}_1 \bar{x}_0 y_1 y_0 + \bar{x}_1 x_0 \bar{y}_1 y_0 + \bar{x}_1 x_0 y_1 \bar{y}_0 + \\
& x_1 \bar{x}_0 \bar{y}_1 \bar{y}_0 + x_1 \bar{x}_0 \bar{y}_1 y_0 + x_1 x_0 \bar{y}_1 \bar{y}_0 + x_1 x_0 y_1 y_0 \\
f_0 =& \bar{x}_1 \bar{x}_0 y_1 \bar{y}_0 + \bar{x}_1 \bar{x}_0 y_1 y_0 + \bar{x}_1 x_0 \bar{y}_1 \bar{y}_0 + \bar{x}_1 x_0 y_1 \bar{y}_0 + \\
& x_1 \bar{x}_0 \bar{y}_1 y_0 + x_1 \bar{x}_0 y_1 y_0 + x_1 x_0 \bar{y}_1 \bar{y}_0 + x_1 x_0 y_1 \bar{y}_0
\end{aligned}
$$

# 2-Bits Adder (module) I





Listing 12: add2bits.v

```
‘timescale 1ns/1ns

module add2bits(
  output f2,
  output f1,
  output f0,
  input  n3,
  input  n2,
  input  n1,
  input  n0
);

  wire n3n, n2n, n1n, n0n;
```

# 2-Bits Adder (module) II

```
notGate not1(n3n, n3);
notGate not2(n2n, n2);
notGate not3(n1n, n1);
notGate not4(n0n, n0);

wire m1, m2, m3, m4, m5, m6, m7, m8, m9, m10,
     m11, m13, m14, m15;

and4Gate and1(m1,   n3n,n2n,n1n,n0 );
and4Gate and2(m2,   n3n,n2n,n1 ,n0n);
and4Gate and3(m3,   n3n,n2n,n1 ,n0 );
and4Gate and4(m4,   n3n,n2 ,n1n,n0n);
and4Gate and5(m5,   n3n,n2 ,n1n,n0 );
and4Gate and6(m6,   n3n,n2 ,n1 ,n0n);
and4Gate and7(m7,   n3n,n2 ,n1 ,n0);
```

# 2-Bits Adder (module) III

```
and4Gate and8(m8,   n3 ,n2n,n1n,n0n);
and4Gate and9(m9,   n3 ,n2n,n1n,n0 );
and4Gate and10(m10, n3 ,n2n,n1 ,n0n);
and4Gate and11(m11, n3 ,n2n,n1 ,n0 );
and4Gate and12(m12, n3 ,n2 ,n1n,n0n);
and4Gate and13(m13, n3 ,n2 ,n1n,n0 );
and4Gate and14(m14, n3 ,n2 ,n1 ,n0n);
and4Gate and15(m15, n3 ,n2 ,n1 ,n0);

wire f2_0, f2_1;
or3Gate o_f20(f2_0,  m7, m10, m11);
or3Gate o_f21(f2_1, m13, m14, m15);
orGate o_f2(f2, f2_0, f2_1);

wire f1_0, f1_1;
```

# 2-Bits Adder (module) IV

```
  or4Gate o_f10(f1_0, m2, m3, m5, m6);
  or4Gate o_f11(f1_1, m8, m9, m12, m15);
  orGate o_f1(f1, f1_0, f1_1);

  wire f0_0, f0_1;
  or4Gate o_f00(f0_0, m1, m3, m4, m6);
  or4Gate o_f01(f0_1, m9, m11, m12, m14);
  orGate  o_f0(f0, f0_0, f0_1);
endmodule
```

# 2-Bit Adder (testbench) I





Listing 13: add2bits_tb.v

```
‘timescale 1ns/1ns

module add2bits_tb;
  logic x0,x1,y0,y1;
  wire f2, f1, f0;

  add2bits add2(f2,f1,f0,x1,x0,y1,y0);

  initial begin
    $dumpfile("add2bits.vcd");
    $dumpvars;

    x1 = 0;
```

# 2-Bit Adder (testbench) II

```
    x0 = 0;
    y1 = 0;
    y0 = 0;

    repeat (2) begin
      repeat (2) begin
        repeat (2) begin
          repeat (2) begin
            #30;
            $display(x1,x0," ",y1,y0," ",
                     f2,f1,f0);
            y0 = y0 + 1;
          end
          y1 = y1 + 1;
        end
```

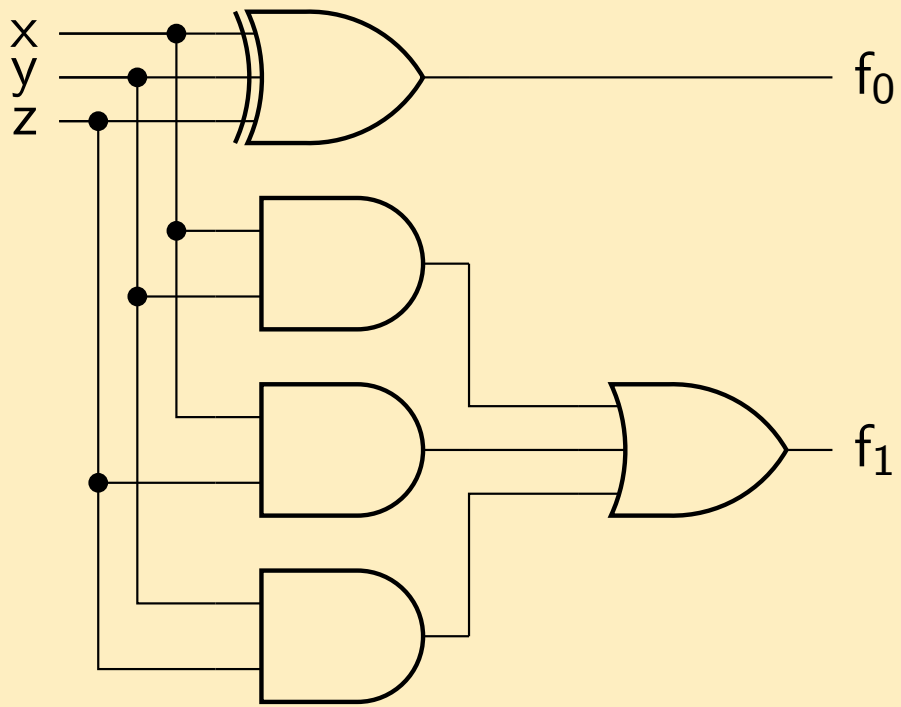
# 2-Bit Adder (testbench) III

```
            x0 = x0 + 1;
        end
        x1 = x1 + 1;
    end
    $finish();
  end

endmodule
```

# Full Adder

(The circuit is from the Logic Gates lecture)

# Full Adder: Simplified Circuit



# fullAdder: Module I



Listing 14: fullAdder.v

```
‘timescale 1ns/1ns

module fullAdder(
  output f1,
  output f0,
  input  x,
  input  y,
  input  z
);

  andGate a1(xy,x,y);
  andGate a2(xz,x,z);
  andGate a3(yz,y,z);
```

# fullAdder: Module II

```
  or3Gate o1(f1,xy,xz,yz);
  xor3Gate x1(f0,x,y,z);
endmodule
```

# Full Adder: Testbench I





Listing 15: fullAdder_tb.v

```
‘timescale 1ns/1ns

module fullAdder_tb;
  wire co, s;
  logic x, y, z;

  fullAdder fa(co,s,x,y,z);


  initial begin
    $dumpfile("fullAdder.vcd");
    $dumpvars;
```

# Full Adder: Testbench II

```
    x = 0;
    y = 0;
    z = 0;

    repeat (2) begin
      repeat (2) begin
        repeat (2) begin
          #20ns;
          $display(x,y,z,co,s);
          z = z + 1;
        end
        y = y + 1;
      end
      x = x + 1;
    end
```

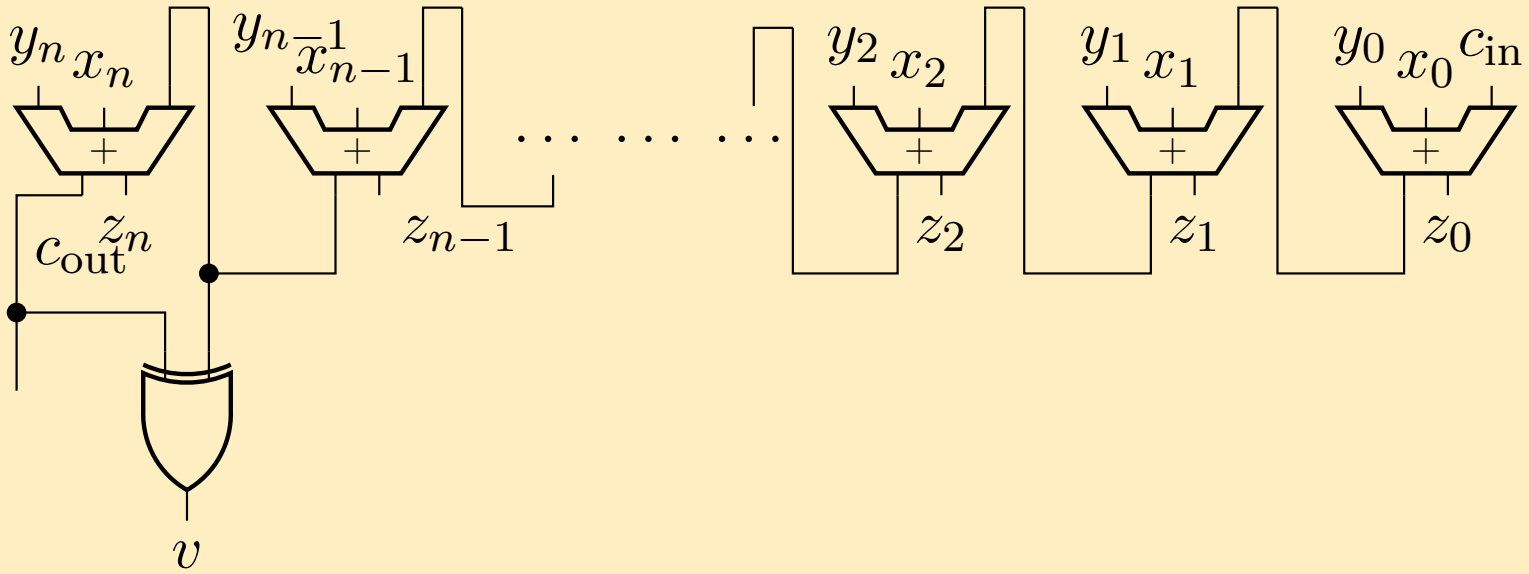
# Full Adder: Testbench III

```
    $finish();
  end

endmodule
```

# Binary Adder

(The circuit is from the Logic Gates lecture)

# Binary Adder: Circuit



# Verilog Features Introduced

- Module parameters
- `parameter` definition
- bus declaration
- bus usage
- `signed`
- `genvar` definition

# Binary Adder (module) I





Listing 16: binaryAdder.v

```
‘timescale 1ns/1ns

module binaryAdder #(parameter WIDTH=1) (
  output cOut,
  output v,
  output [WIDTH-1:0] sum,
  input  [WIDTH-1:0] x,
  input  [WIDTH-1:0] y,
  input cIn
);
  wire [WIDTH-1:0] c;

  assign cOut = c[WIDTH-1];
```

# Binary Adder (module) II

```
generate
  if (WIDTH > 1)
    xorGate xg(v,c[WIDTH-1], c[WIDTH-2]);
  else
    xorGate xg(v,c[WIDTH-1], cIn);
endgenerate

fullAdder fa(c[0],sum[0],x[0],y[0],cIn);

for (genvar i = 1; i < WIDTH; i = i + 1)
begin
  fullAdder fa(c[i],sum[i],x[i],y[i],
               c[i-1]);
end
```

# Binary Adder (module) III

```
endmodule
```

# Binary Adder: Testbench I



Listing 17: binaryAdder_tb.v

```
`timescale 1ns/1ns

module binaryAdder_tb;
  parameter N = 8;

  wire c,v;
  wire [N-1:0] sum;
  logic [N-1:0] x, y;

  binaryAdder #(N) ba(c, v, sum, x, y, 1'b0);

  initial begin
    logic signed [N-1:0] sums, xs, ys;
```

# Binary Adder: Testbench II





```
$dumpfile("binaryAdder.vcd");
$dumpvars;

repeat (100) begin
   x = $urandom_range(0,2**N-1);
   y = $urandom_range(0,2**N-1);
   #60ns;
   xs = x;
   ys = y;
   sums = sum;
   $display("%d %d %d %d ** %d %d %d %d",
      x, y, sum, c,
      xs, ys, sums, v);
end
```

# Binary Adder: Testbench III

```
    $finish();
  end

endmodule
```

(Carmi) Lecture 9 reached here