CSE-308: Digital System Design

Lecture 1

Introduction

warm-up

quiz…

DLD (also DSD) is basically a design course.

List all the steps involved in capturing a real-world phenomenon and converting it to a simplest digital logic circuit using discrete gates.

Note: You just have to list steps. Don’t write unnecessary details.

Instructor:

Rehmat Ullah Khattak

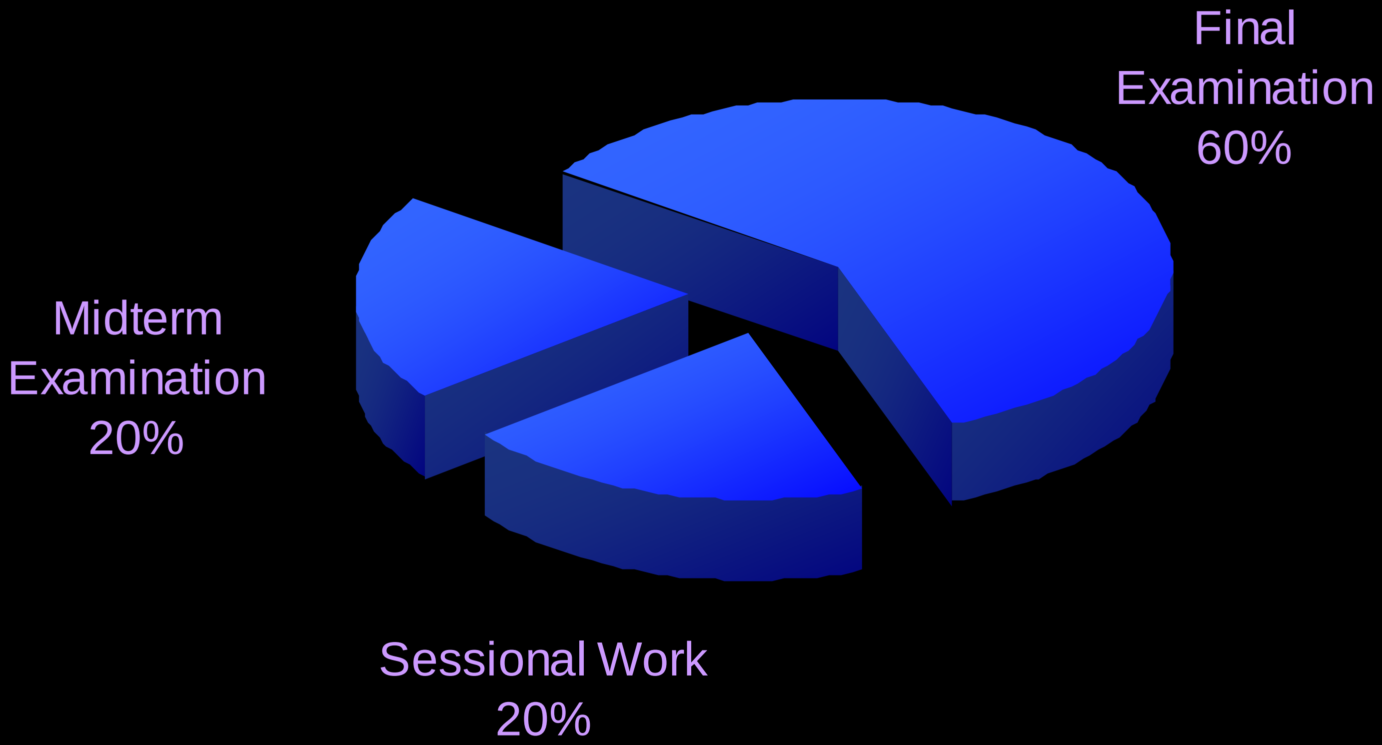
rehmatullah@uetpeshawar.edu.pk

Course Newsgroup:

https://classroom.google.com/c/NDcxNjg0Mzc2ODI0?cjc=g3ofb5b

Check the group often for announcements related to the course marks

distribution…


Final Examination 60%
Midterm Examination 20%
Sessional Work 20%

Sessional Work (20%)

- Homeworks 5%
- Quizzes 5%
  - (50% announced + 50% unannounced)
- Class Participation + Attendance 10%

Homeworks (5%)

- Almost one every week
- Late homework policy: Late submission of homeworks will result in "zero" grade

Homeworks (contd…)

- Done on an individual basis
- Collaboration is fine, but it should be you alone who writes up the answers
- Copying of homework is allowed as long as it goes through the head

Midterm Exam (20%)

- During the 8th week
- Duration: Two hours
- Will cover all material covered during the first 8 weeks

Final Exam (50%)

- During the 17th week
- Will cover the whole of the course before and after midterm
- Duration: Two/three hours

What I don’t want?

That you fail such an easy course

Essential ingredient to pass is

"HARD WORK"

attendance

policy…

Students with attendance less than 75%

will NOT be allowed to sit in the

"EXAM"

© CET
I.I.T. KGP

Course Outline

1. Digital circuit design flow
2. Verilog Hardware Description Language
3. Logic Synthesis
   – Multilevel logic minimization
   – Technology mapping
   – High-level synthesis
4. Testability Issues
5. Physical Design Automation
   – Floorplanning, placement, routing, etc.

References

1. Contemporary logic design
   R.H. Katz, Addison-Wesley Publishing Co., 1993.
2. Application-specific integrated circuits
   M.J.S. Smith, Addison-Wesley Publishing Co., 1997.
3. Modern VLSI design: systems on silicon
   W. Wolf, Pearson Education, 1998.
4. Verilog HDL synthesis: a practical primer
   J. Bhasker, BS Publications, 1998.
5. High-level synthesis: introduction to chip and system design
   D.D. Gajski, N.D. Dutt, A.C. Wu and A.Y. Yin, Kluwer Academic Publishers, 1992.

6. **Digital systems testing and testable design**
   M. Abramovici, M.A. Breuer and A.D. Friedman, IEEE Press, 1994.
7. **Built-in test for VLSI: pseudo-random techniques**
   P. Bardell, W.H. McAnney and J. Savir, J. Wiley & Sons, 1987.
8. **An introduction to physical design**
   M. Sarrafzadeh and C.K. Wong, McGraw Hill, 1996.
9. **Algorithms for VLSI physical design automation, 3rd Edition**
   N.A. Sherwani, Kluwer Academic Publishers, 1999.
10. **VLSI physical design automation: theory and practice**
   S.M. Sait and H. Youssef, World Scientific Pub. Co., 1999.

© CET
I.I.T. KGP

Digital Circuit Design Flow

Digital Design Process

© CET
I.I.T. KGP

- **Design complexity increasing rapidly**
  - Increased size and complexity
  - CAD tools are essential
- **The present trend**
  - Standardize the *design flow*

What is design flow?

- Standardized design procedure
  - Starting from the design idea down to the actual implementation
- Encompasses many steps
  - Specification
  - Synthesis
  - Simulation
  - Layout
  - Testability analysis
  - Many more ……

ASIC
FPGA

- **New CAD tools**
  - Based on *Hardware description language* (HDL)
  - HDLs provide formats for representing the outputs of various design steps
  - An HDL based CAD tool transforms from its HDL input into a HDL output which contains more hardware information.
    - Behavioral level to register transfer level
    - Register transfer level to gate level
    - Gate level to transistor level

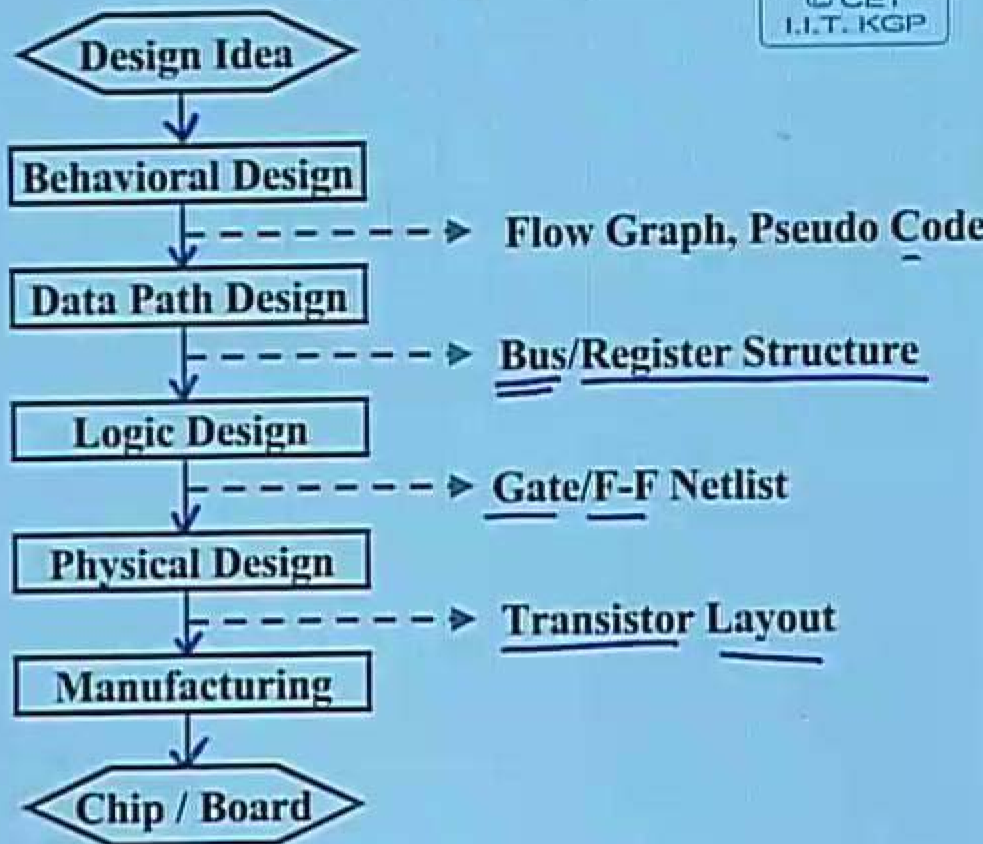
Two Competing HDLs

© CET I.I.T. KGP

1. Verilog
2. VHDL

In this course we would be concentrating on <u>Verilog</u> only

Simplistic View of Design Flow

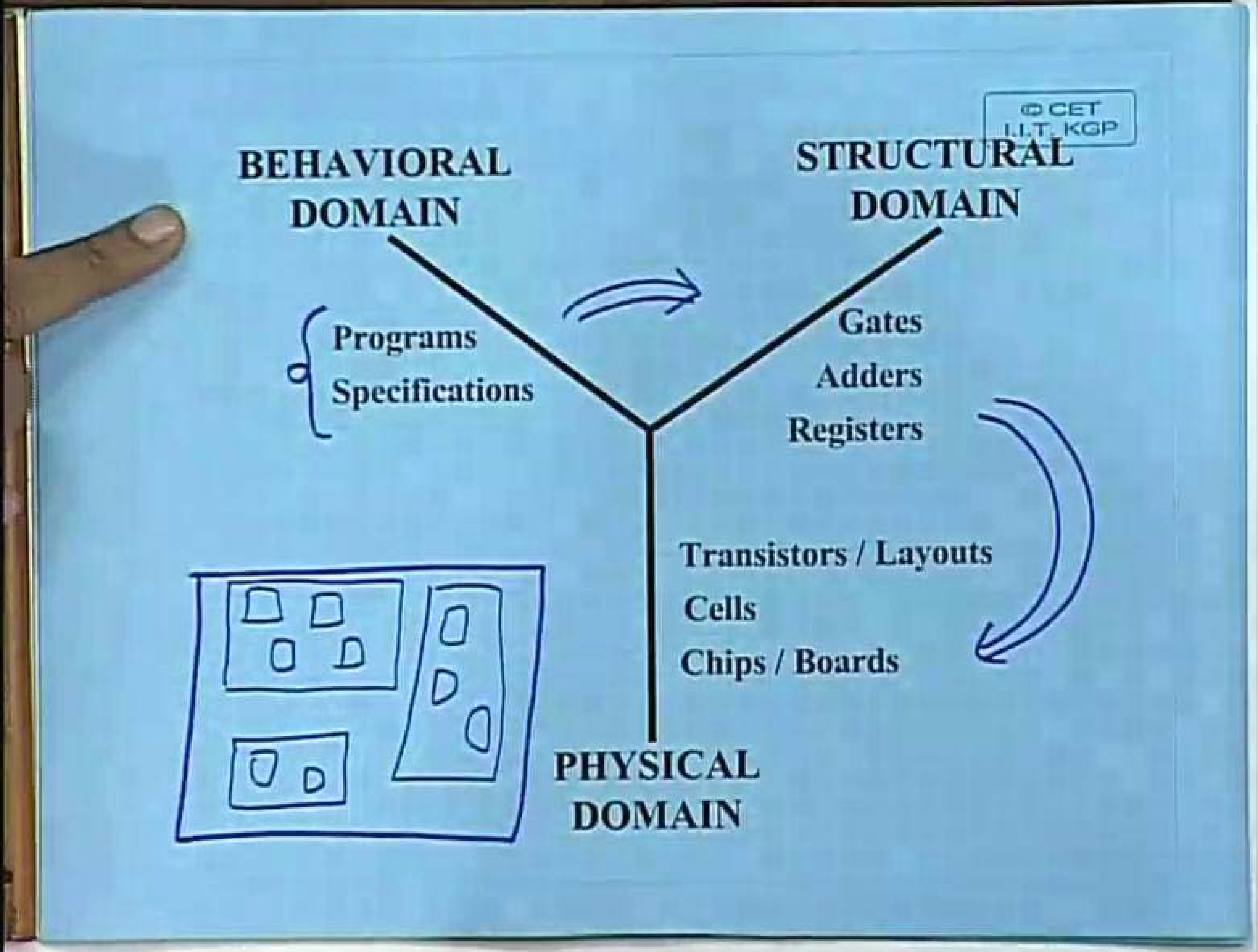


Design Representation

© CET
I.I.T. KGP

- A design can be represented at various levels from three different angles:
    1. Behavioral
    2. Structural
    3. Physical

- Can be represented by Y-diagram



© CET I.I.T. KGP
BEHAVIORAL DOMAIN
STRUCTURAL DOMAIN
Programs
Specifications
Gates
Adders
Registers
Transistors / Layouts
Cells
Chips / Boards
PHYSICAL DOMAIN

Behavioral Representation

© CET I.I.T. KGP

- Specifies how a particular design should respond to a given set of inputs.
- May be specified by
  - Boolean equations
  - Tables of input and output values
  - Algorithms written in standard HLL like C
  - Algorithms written in special HDL like Verilog

An algorithmic level description of Cy

```
module  carry (cy, a, b, c);
      input a, b, c;
      output cy;
      assign
         cy = (a&b) | (b&c) | (c&a);
 endmodule
```

Boolean behavioral specification for Cy

```
primitive  carry (cy, a, b, c);
        input a, b, c;
        output cy;
        table
        // a  b  c     cy
           1  1  ?  :  1 ;
           1  ?  1  :  1 ;
           ?  1  1  :  1 ;
           0  0  ?  :  0 ;
           0  ?  0  :  0 ;
           ?  0  0  :  0 ;
      endtable
endprimitive
```

comment

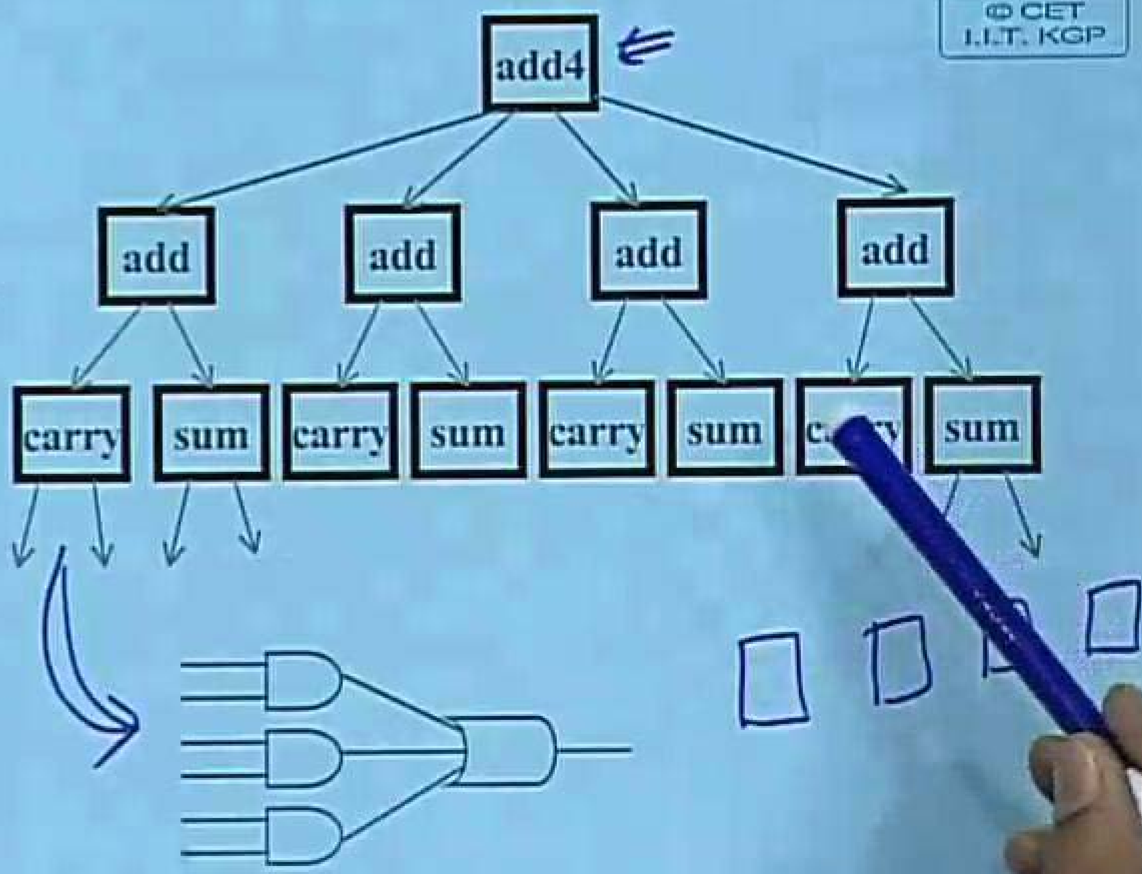
000

111

Structural Representation

- Specifies how components are interconnected.
- In general, the description is a list of modules and their interconnects.
  - called ***netlist.***
  - can be specified at various levels.

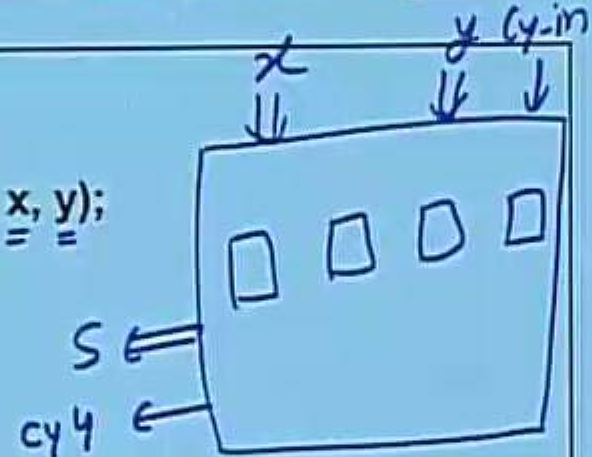
- At the structural level, the levels of abstraction are
  - the module level
  - the gate level
  - the switch level $\Rightarrow$ transistor
  - the circuit level
- In each level more detail is revealed about the implementation.


add4
add
add
add
add
carry
sum
carry
sum
carry
sum
sum

Structural representation :: example

4-bit adder

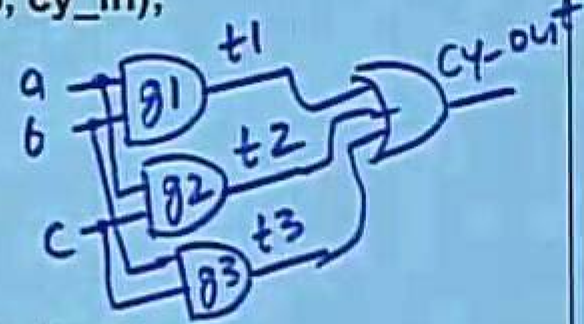
```
module add4 (s, cy4, cy_in, x, y);
        input [3:0] x, y;
        input cy_in;
        output [3:0] s;
        output cy4;
        wire [2:0] cy_out;
            add B0 (cy_out[0], s[0], x[0], y[0], ci);
            add B1 (cy_out[1], s[1], x[1], y[1], cy_out[0]);
            add B2 (cy_out[2], s[2], x[2], y[2], cy_out[1]);
            add B3 (cy4, s[3], x[3], y[3], cy_out[2]);
endmodule
```



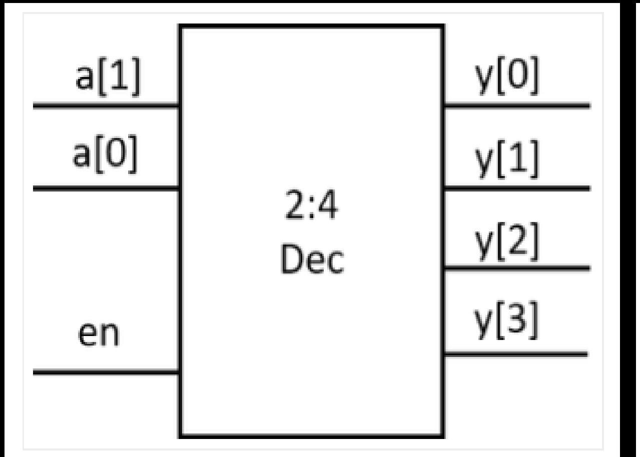
```
module  add  (cy_out, sum, a, b, cy_in);
    input a, b, cy_in;
    output  sum, cy_out;
        sum   s1 (sum, a, b, cy_in);
        carry  c1 (cy_out, a, b, cy_in);
 endmodule
```

Instantiate

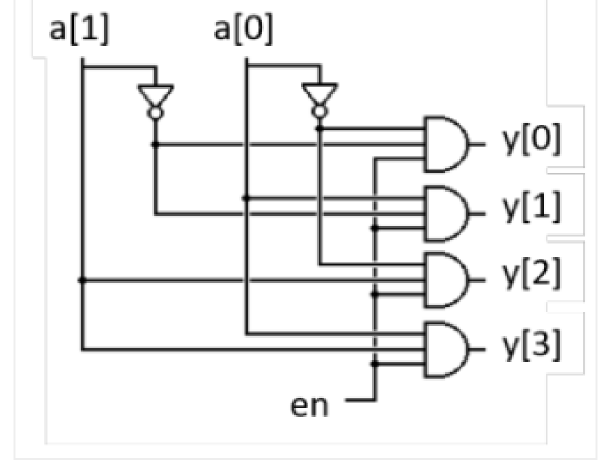
```
module carry (cy_out, a, b, cy_in);
    input a, b, cy_in;
    output  cy_out;
    wire  t1, t2, t3;
        and  g1 (t1, a, b);
        and  g2 (t2, a, c);
        and  g3 (t3, b, c);
        or    g4 (cy_out, t1, t2, t3);
 endmodule
```



2x4 Decoder



2:4 Decoder Block diagram



2:4 Decoder Schematic Diagram

Inputs			Outputs			
en	a[1]	a[0]	y[3]	y[2]	y[1]	y[0]
0	×	×	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

2:4 Decoder Truth Table

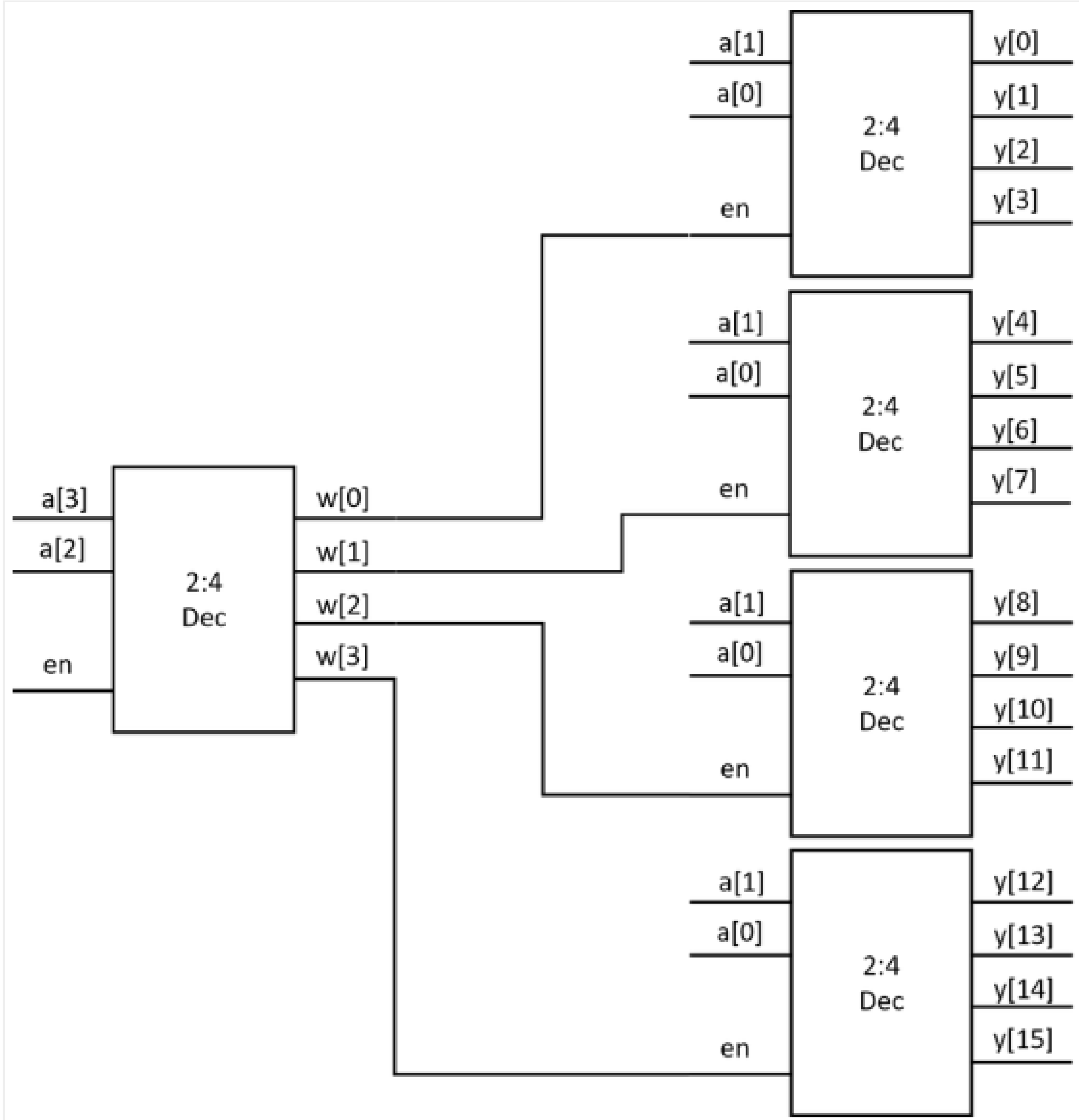
2x4 Decoder Structural/Gate Level Modeling

```
module dec24_str(

output [3:0] y,

input [1:0] a,

input en);

and (y[0], ~a[1], ~a[0], en); /* 3-input AND gates */

and (y[1], ~a[1], a[0], en);

and (y[2], a[1], ~a[0], en);

and (y[3], a[1], a[0], en);

endmodule
```

2x4 Decoder in Behavioral Modeling using a case statement

```
module dec24_beh(
output reg [3:0] y,
input [1:0] a,
input en);
always @(*)
if(en) /* only if en = 1, case statement will execute */
case(a)
0: y = 4'b0001;
1: y = 4'b0010;
2: y = 4'b0100;
3: y = 4'b1000;
default: y = 0;
endcase
else y = 0; /* if en = 0, all bits of y will remain zero */
endmodule
```

4x16 Decoder using five 2x4 Decoders

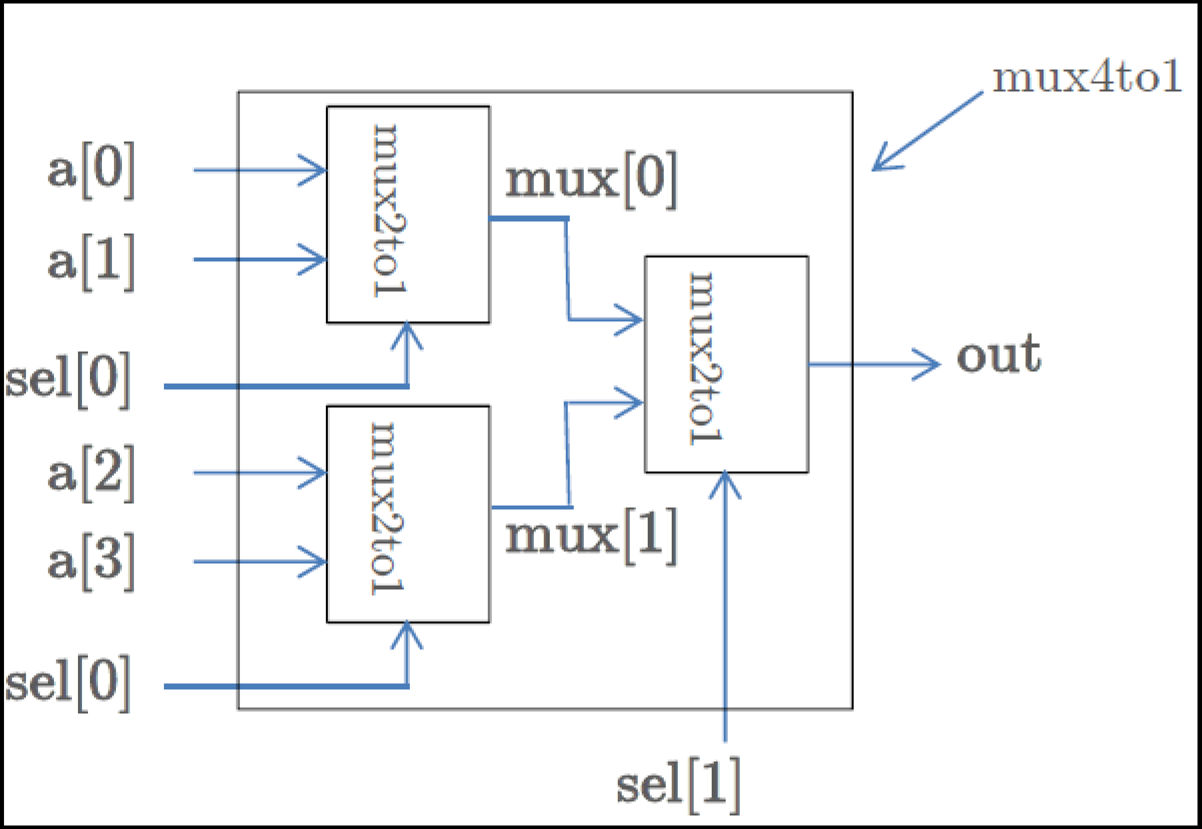


Schematic of 4:16 decoder using five 2:4 decoders

4x16 Decoder using five 2x4 Decoders

```
module dec4x16_str(
output [15:0] y,
input [3:0] a,
input en
);
wire [3:0] w;
dec2x4_str u0(w, a[3:2], en);
dec2x4_str u1(y[3:0], a[1:0], w[0]);
dec2x4_str u2(y[7:4], a[1:0], w[1]);
dec2x4_str u3(y[11:8], a[1:0], w[2]);
dec2x4_str u4(y[15:12], a[1:0], w[3]);
endmodule
```

4x1 Multiplexer using 2x1 Multiplexers

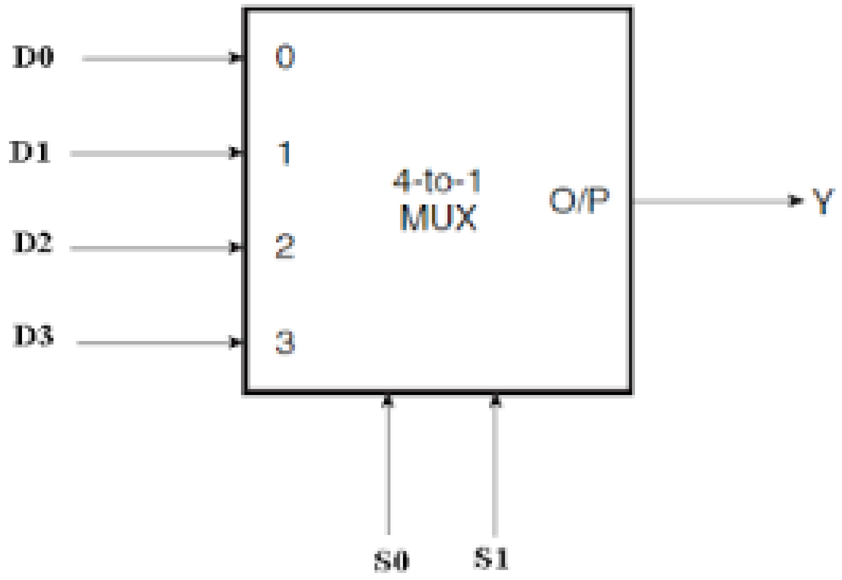


4x1 Multiplexer using 2x1 Multiplexers

```
module mux2to1(a,b,sel,out);
        input a,b,sel;
        output out;
        assign out = sel?b:a;
endmodule
```

```
module mux4to1(a,sel,out);
        input [3:0] a;
        input [1:0] sel;
        output out;
        wire mux[1:0];

        mux2to1 m1 (a[0],a[1],sel[0],mux[0]),
                m2 (a[2],a[3],sel[0],mux[1]),
                m3 (mux[0],mux[1],sel[1],out);
endmodule
```

4x1 Multiplexer



X_1	X_0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

And the equation is

$$Y = D0\,\overline{S1}\,\overline{S0} + D1\,\overline{S1}\,S0 + D2\,S1\,\overline{S0} + D3\,S1\,S0$$

Gate Level Modeling

```
module gatelevel(i0,i1,i2,i3,s0,s1,y);

input i0,i1,i2,i3,s0,s1;

output y;

wire a,b,c,d,e,f;      // for partial outputs

not (a,s0);            // a=~s0

not (b,s1);

and (c,i0,a,b);        // c=i0&&a&&b

and (d,i1,s0,b);

and (e,i2,a,s1);

and (f,i3,s0,s1);

or (y,c,d,e,f);       // y=c||d||e||f

endmodule
```

Behavioral Modeling

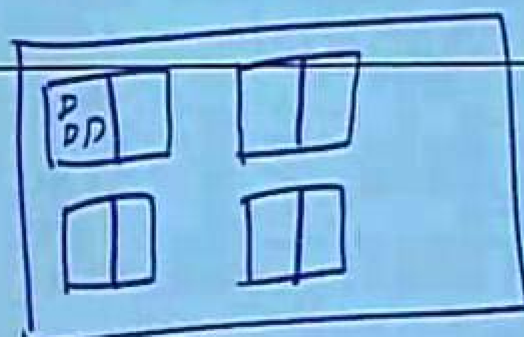
```
module behavioral(

input i0,i1,i2,i3,

output reg y,

input[1:0] sel

);

always @ (i0,i1,i2,i3,sel)   // Mention all the inputs inside the always block

case(sel)

2'b00: y=i0;         // Follow the truth table

2'b01: y=i1;

2'b10: y=i2;

2'b11: y=i3;

endcase

endmodule
```

Data Flow Modeling

```
module dataflow(i0,i1,i2,i3,s0,s1,y);

input i0,i1,i2,i3,s0,s1;

output y;

assign y=(!s0&&!s1&&i0)||(s0&&!s1&&i0)||(!s0&&s1&&i2)||(s0&&s1&&i3);

endmodule
```

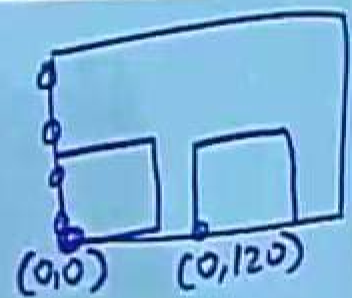
Physical Representation

- The lowest level of physical specification.
  - Photo-mask information required by the various processing steps in the fabrication process.
- At the module level, the physical layout for the 4-bit adder may be defined by a rectangle or polygon, and a collection of ports.



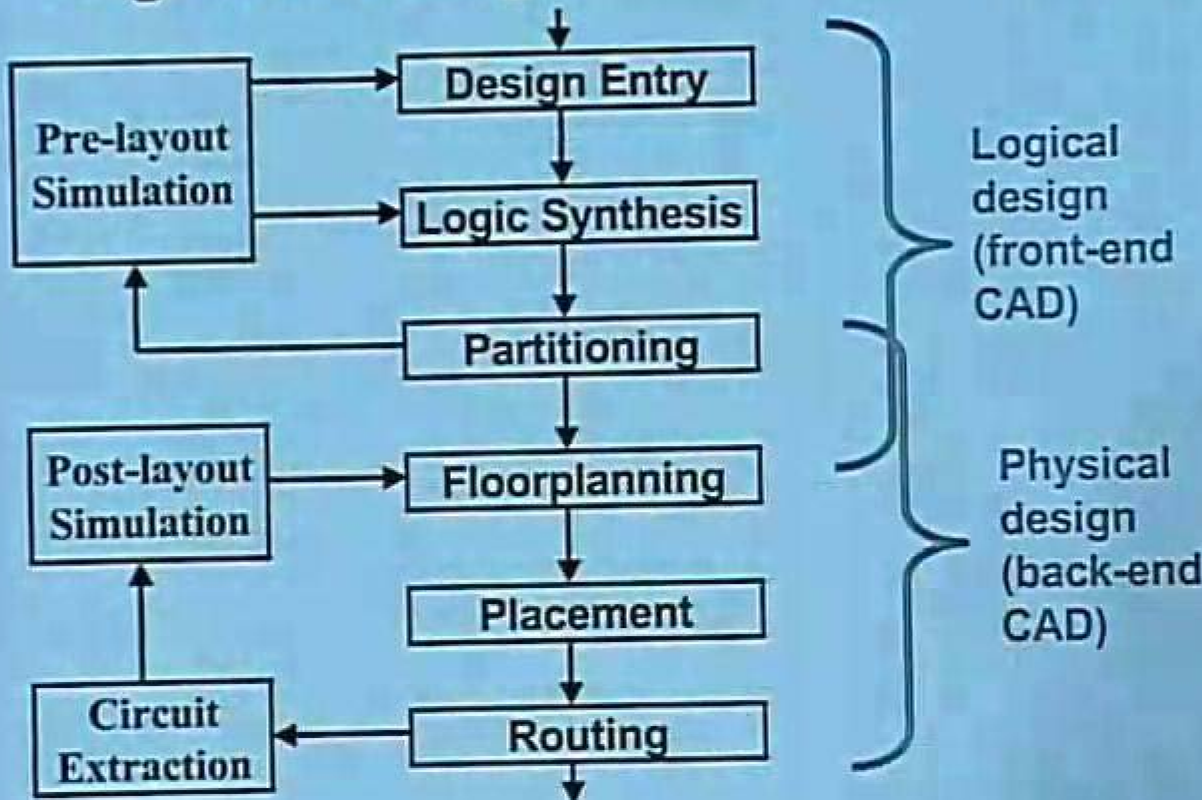
Physical representation -- example

Imaginary physical description for 4-bit adder

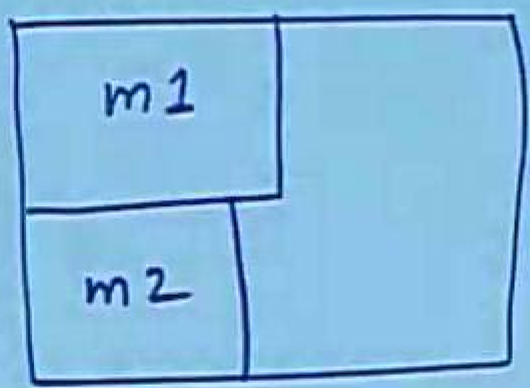


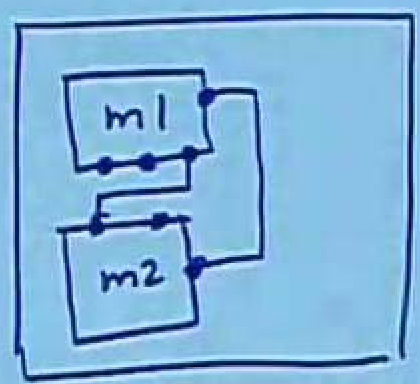

```
module add4( - -- ----    )
    input x[3:0], y[3:0];
    input cy_in;
    output s[3:0];
    output cy4;
    boundary [0, 0, 130, 500];
    port x[0]     aluminum  width = 1 origin = [0, 35];
    port y[0]     aluminum  width = 1 origin = [0, 85];
    port  cy_in  polysilicon width = 2 origin = [70, 0];
    port s[0]     aluminum  width = 1  origin = [120, 65];

    add a0 origin  =  [0, 0];
    add a1 origin  =  [0, 120];

endmodule
```

Digital IC Design Flow: A quick look



k look





Logical design (front-end CAD)

Physical design (back-end CAD)