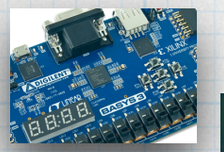
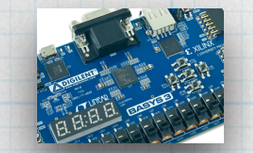


A Stratix IV FPGA from Altera

Hardware Synthesis Lab

Krerk Piromsopa, Ph.D. and Arthit Thongtuk, D.eng

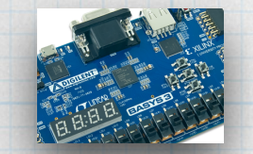


Instructors

Krerk Piromsopa, Ph.D.

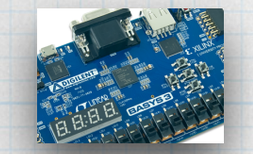
Arthit Thongtuk, D.Eng.

We have TAs.



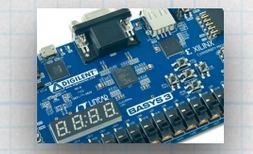
Rules.

* Always watch video and prepare labs from home.
* There might be a quiz at the beginning of the class. Don't be late.
* Discuss issues on the facebook page.
* Take good care of your devices.
  (You may have to pay if you break it.)
* Unless stated otherwise, each lab is an individual assignment.
* No cheating. Do not copy from your friends or any web page.
* Project will be announced later.
* Should you have any further question, talk to you instructors.



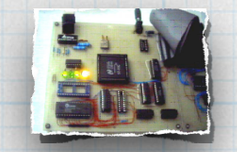
(Tentative) Schedule

Week	Description	Student Assignment
1	Introduction to VerilogHDL & Hardware Synthesis tools	Software Installation
2	Lecture: TDM & Design Contraint Lab: Simple FSM simulation Quiz: Test bench	
3	Lecture: Serial Communication Lab: Counter Quiz: TDM, Single Pulser, Counter	
4	Lecture: Memory and ROM Lab: Serial Communication Quiz: Serial Protocol	
5	Lecture: N/A Lab: Simple CPU Quiz: Memory, ROM, CPU	
6	skill test	
7	Lecture: External I/O Lab: Multiple-cycle CPU Quiz: N/A	
8	Lecture: System on a Chip Lab: VGA, USB Quiz: VGA signal, USB, I/O	
9	Lecture: N/A Lab: Pipelining Processor Quiz: Pipeline Design	
10	Lecture: N/A Lab: Vector Processor Quiz: Vector Design	
11	Lecture: Project Assignment Lab: Cache Quiz: Cache Design	
12	Lecture: N/A Lab: MMU Quiz: Virtual Memory Design	
13-14	Lecture: N/A Lab: N/A (Project) Quiz: TBA	
15	Project submission	



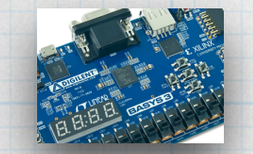
Grading

* 50% Laboratories
* 20% Quiz
* 30% Project
* If you cannot get to the lab for any reason, please contact your instructors.



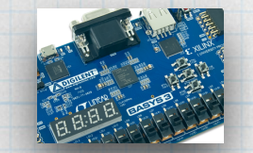
Introduction to VerilogHDL and Digital Synthesis

Krerk Piromsopa, Ph.D.
Computer Engineering, Chulalongkorn University



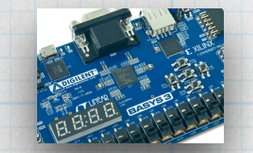
Outline

* Introduction to VerilogHDL
  * (Brief) History of HDL
  * VerilogHDL, Tip & Trick, Pitfalls, Fallacies, Good designs
* Digital Synthesis Tools
* Basic Digital Simulation



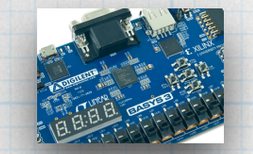
"After this class, if you have a better idea of using Verilog as a language to describe your design,
I have succeed."

Our goals for today



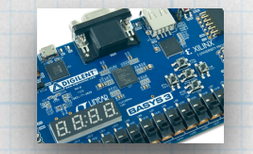
What is HDL?

Hardware Description Language



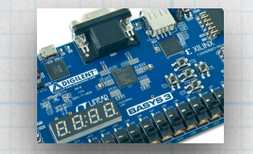
Hardware Description Language

* Special Computer Language to describe structure, design and operation of electronic (digital logic) circuits
* Specification, Modeling languages
    * e.g. Timing, Design, Connection
* Design is not a program



Life without HDL

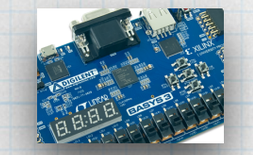
* Boolean functions / equations
* Logic Gates
* Timing Diagram
* State Diagram
* ASM Chart



Life without HDL

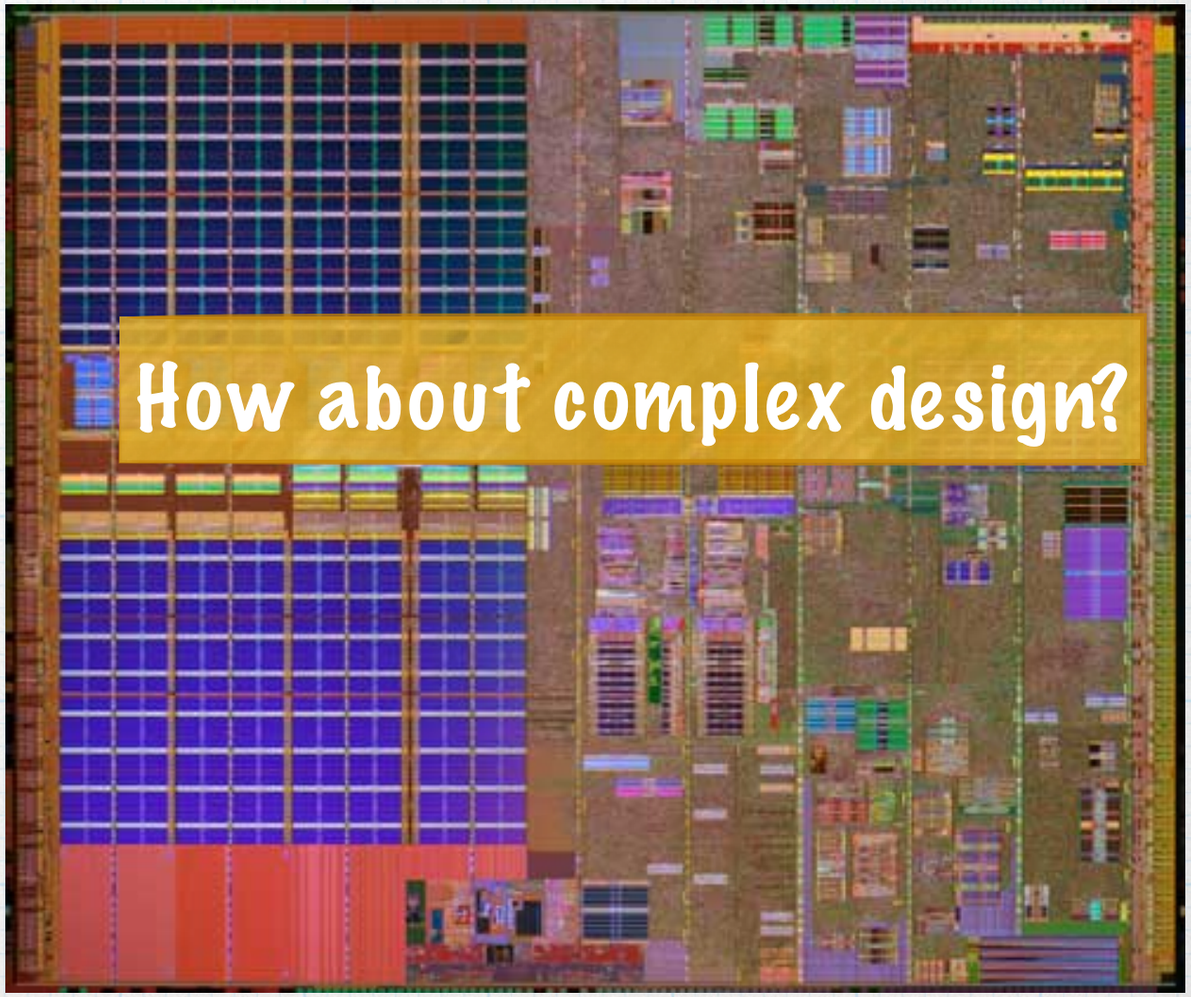
* Boolean functions / equations
* Logic Gates
* Timing Diagram
* State Diagram
* ASM Chart

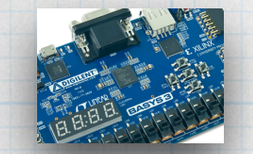
How about complex design?



Life without HDL

* Boolean functions / equations
* Logic Gates
* Timing Diagram
* State Diagram
* ASM Chart





Evolution of HDL

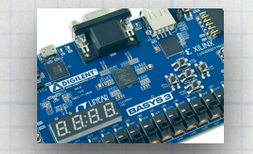
* Historically: ABEL HDL, AHDL, UPD, JHDL, PALASM, VHDL, VerilogHDL

* Currently: Two main languages

  * VHDL (for documentation)
    1983, influenced by ADA, PASCAL

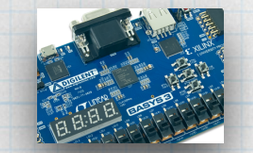
  * VerilogHDL (for simulation)
    1983, influenced by C, FORTRAN

* Synthesis to Hardware

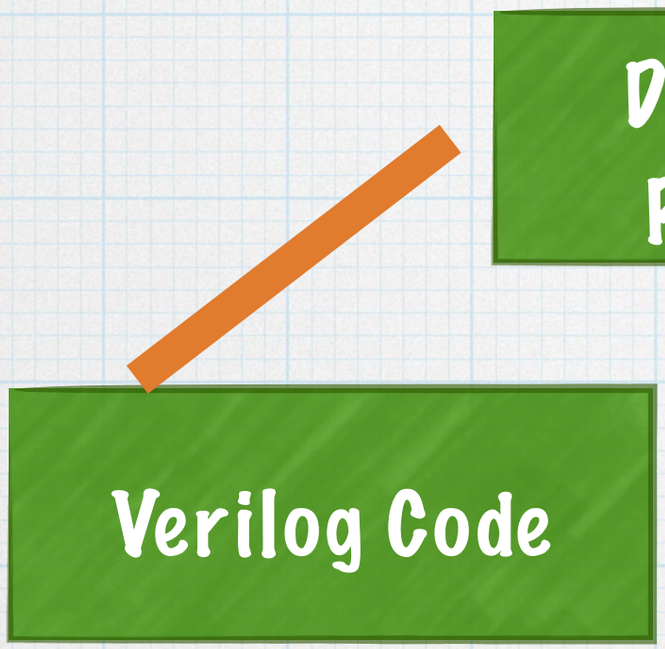


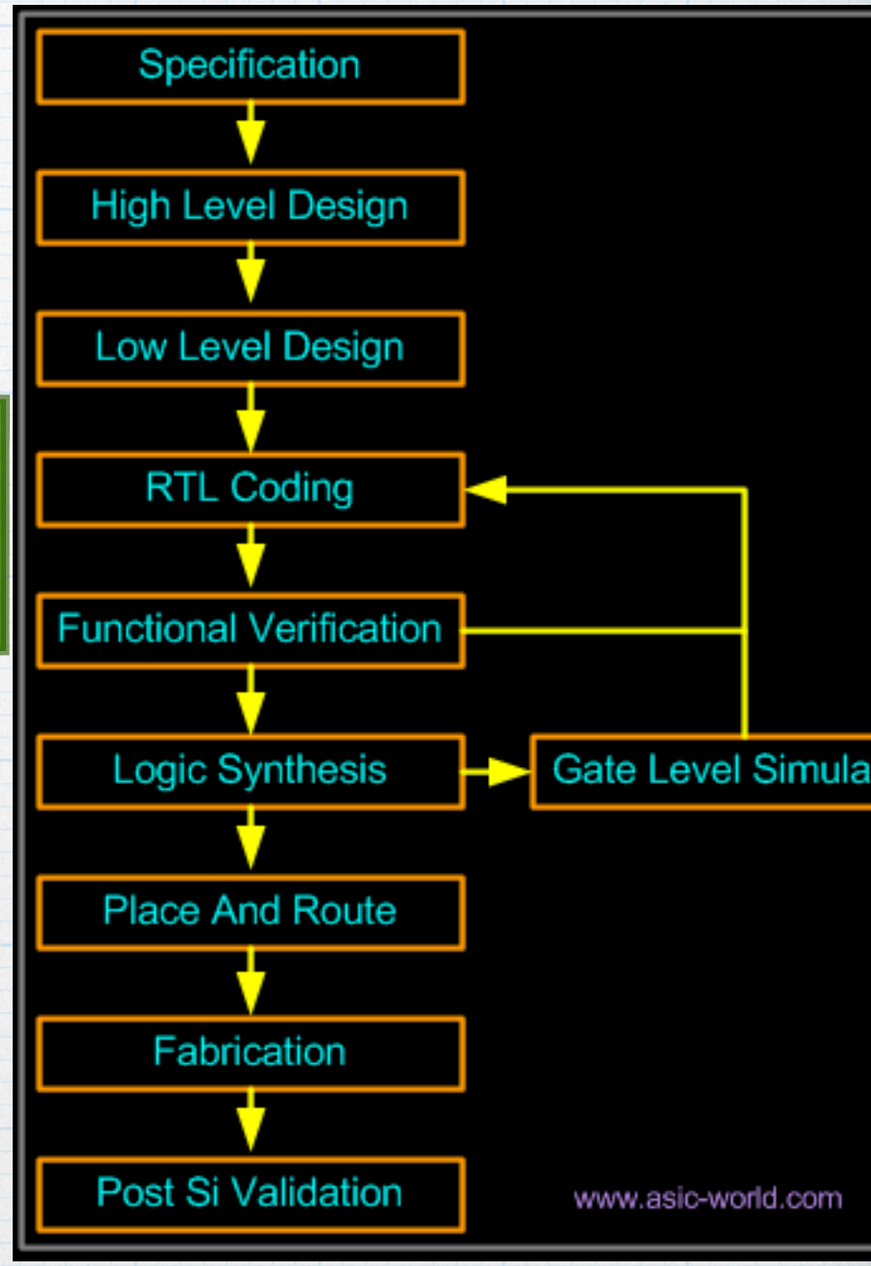
VerilogHDL

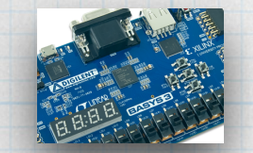
* Invented by Phil Moorby in 1984 by combining HiLo (contemporary HDL) and C
* Becoming standard in 1995 (IEEE Verily standard 1364)
* Verilog-2001 is a revision with simplified syntax. (2003)
* Verilog-2005 (minor corrections)



Design Flow

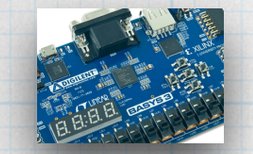

Design and Partition
Verilog Code
Test Case
Simulate
Synthesis


Specification
High Level Design
Low Level Design
RTL Coding
Functional Verification
Logic Synthesis
Gate Level Simula
Place And Route
Fabrication
Post Si Validation
www.asic-world.com



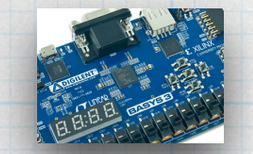
VerilogHDL - Language

* Logic Values
  * 0, 1 , x (unknown),
    z (high impedance)
* number representation
  * 6'b10_011, 16'hx, 6'o77, 10'd44
  * 6'b_101111 WRONG



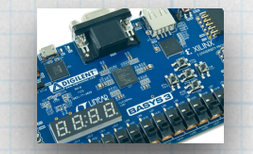
Syntax

* C-style syntax
  * case sensitive
  * Comment // (line), /* */ (block)
  * Use ; (semicolon) to end statement
  * Use begin .. end for compound statement



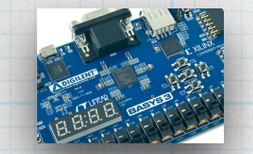
Identifier

* Not a variable
* Specify wires (net), registers, or components (modules)
* type [bits] name [dimension]
  * e.g.
    wire [3:0] dat [1:5]



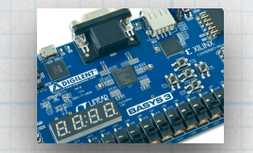
Wires, Registers

* (wire) Wire/Net cannot hold value
* (reg) Registers can hold value
  * This is not a real flipflop.



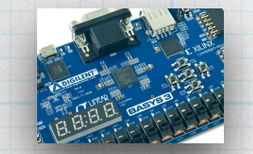
Basic

* Parameter is a constant, can only be integer
  * parameter WIDTH = 32, depth = 1024;
* Nets, Registers
* bitwise and logic
* comparison ==, === (only w===v)
* case, casex

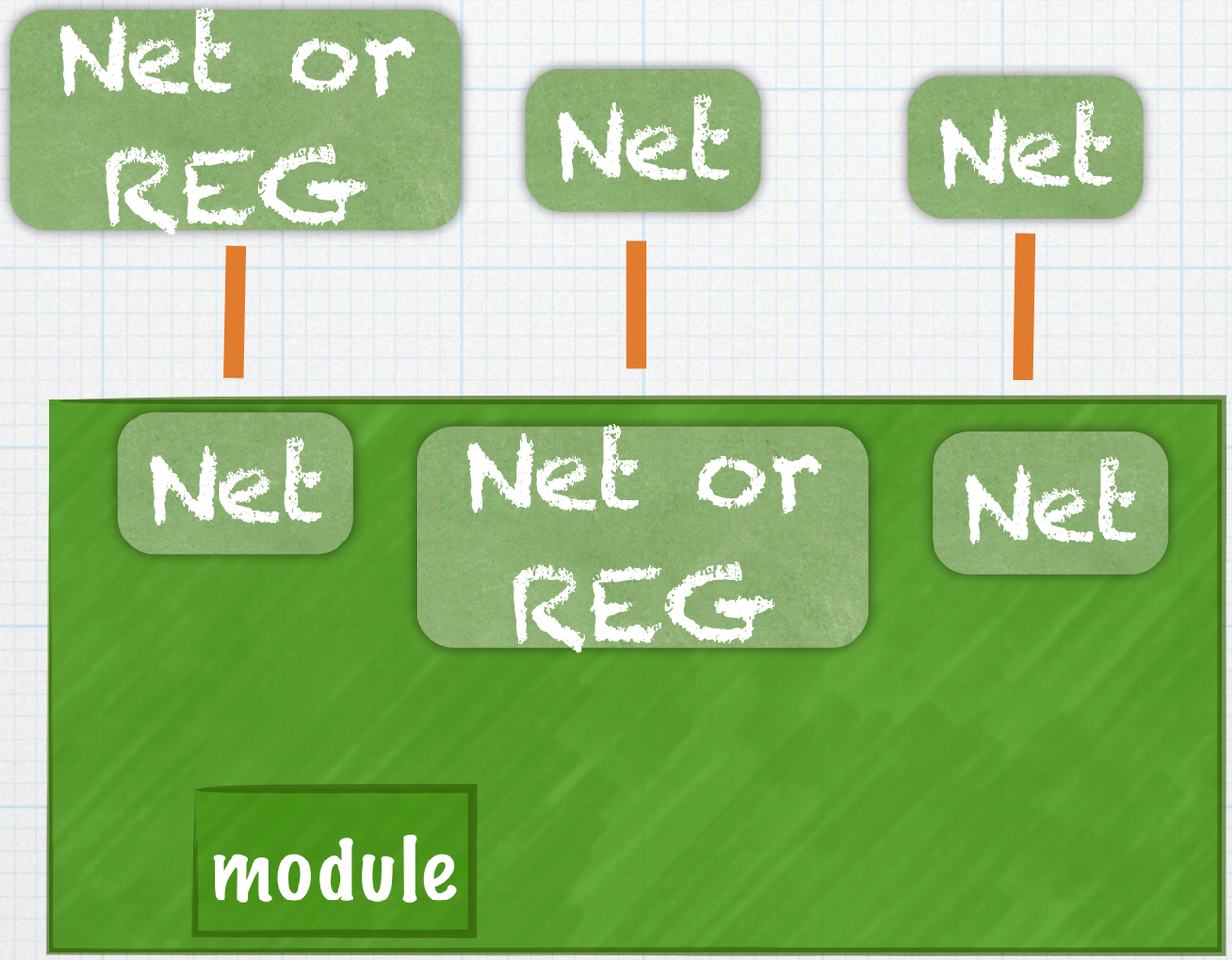


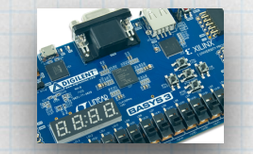
Wire Grouping

- Concatenation
  - { cout, sum }
- Replication
  - {8{2'b01}};



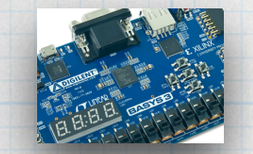
Module Interconnection





Selection

* Variable Part-Select (Verilog 2001)
  * sum[K+:3] --- {sum[K+2],sum[K+1],sum[K]}
* Memories
  * parameter WIDTH = 32, depth = 1024
  * reg [WIDTH-1:0] cache [0:depth-1];
  * reg [WIDTH-1:0] cache [0: 2**WIDTH -1]; // (Verilog 2001)



Module (Template)

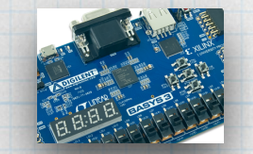
* Verilog 1995

```
module add  (sum, c_out, a, b);

output sum, c_out;

input a,b;

endmodule
```

* Verilog 2001

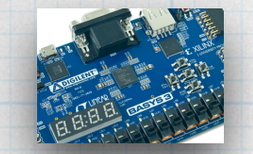
```
module add(output sum, c_out, input a,b);

endmodule
```

* Notion/Practice
  * Output before input



Assignment

* Continuous assignment
  * a = b & c;
    * left-handed side must be net.
* Procedural assignment
  * in always
  * left-handed size must be reg.



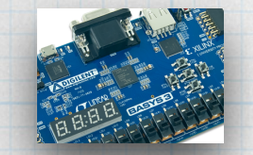
Event

* always @ (events) begin

  end

  * events are synchronization signals or all sensitivity list.

  * posedge , negedge



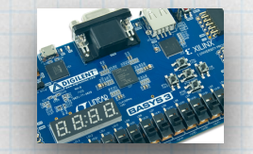
Block

* begin .... end (sequential)
* fork ... join (parallel)

```
begin
  a = 1;
  #10 a = 0;
  #5 a = 4;
end
```

```
fork
  a = 1;
  @(b);
  a = 0;
join
```

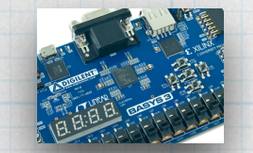
```
fork
  a = 1;
  #10 a = 0;
  #5 a = 4;
join
```



Primitives

* and
* nand
* or
* nor
* xor
* xnor
* not

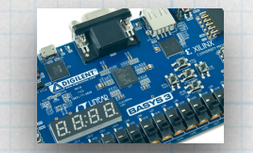
```
nand #(1,2) gate1 (y, i1, i2);
```



User-defined Primitives (UDPs)

```
primitive latch(q_out,en, data);

output reg q_out;

input en, data;

table

// en data : state : q_out/next_state

1 1 : ? : 1;

1 0 : ? : 0;

0 ? : ? : -;

x 0 : 0 : -;

x 1 : 1 : -;


endtable
```

* Primitives
* Common Components (e.g. gates)
* Put truth table into the code



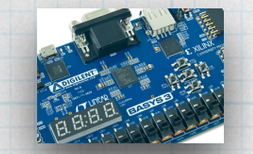
Case, CaseX

```
reg [1:0] address;
case (address)
  2'b00 : statement1;
  2'b01, 2'b10 : statement2;
  default : statement3;
endcase
```

```
reg a;
case (a)
  1'b0 : statement1;
  1'b1 : statement2;
  1'bx : statement3;
  1'bz : statement4;
endcase
```

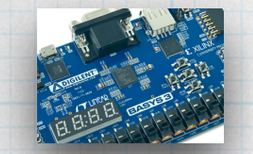
```
reg a;
casez (a)
  1'b0 : statement1;
  1'b1 : statement2;
  1'bx : statement3;
  1'bz : statement4;
endcase
```

```
reg a;
casex (a)
  1'b0 : statement1;
  1'b1 : statement2;
  1'bx : statement3;
  1'bz : statement4;
endcase
```



(System) Tasks

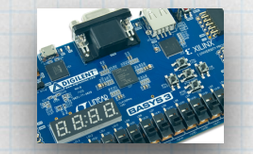
- $display l $displayb l $displayh l $displayo (arguments) ;
- $write l $writeb l $writeh l $writeo (arguments) ;
- $strobe l $strobeb l $strobeh l $strobeo (arguments) ;
- $monitor l $monitorb l $monitorh l $monitoro (arguments) ;



Override Parameters

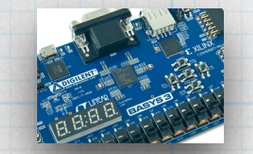
```
module adder (sum, c_out,a,b);
parameter width=4;
output reg [width-1:0] sum;
output reg c_out;
input [width-1:0] a,b;
always @(a,b) begin
{c_out,sum} = a+b;
end
endmodule
```

```
module top();

....
// 8 bit adder
adder  #(8) add8(.....)
// 16 bit adder
adder #(16) add16 (...)


endmodule
```



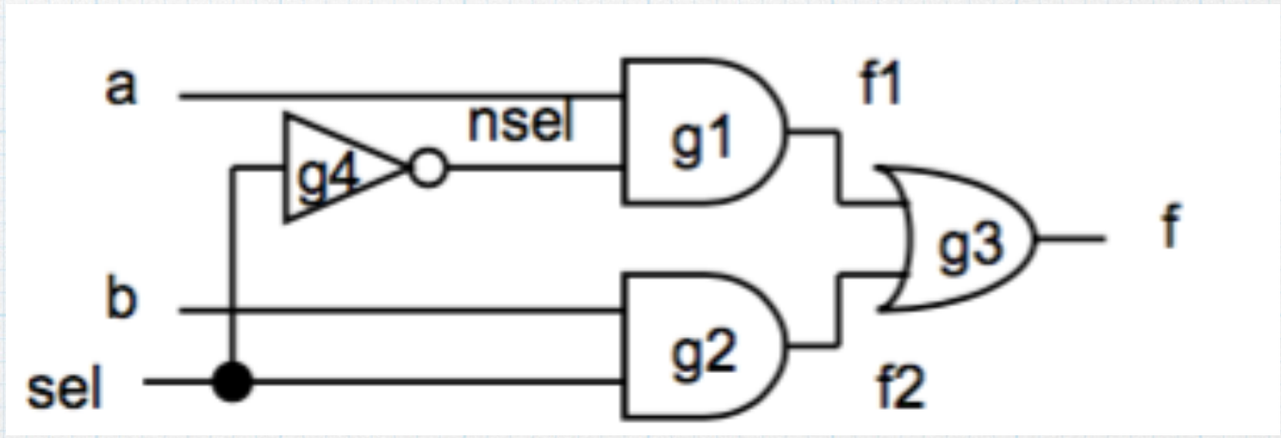
Level of Abstractions

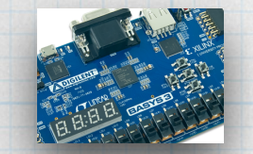
* Gate Level
    * Use Primitives
* Behavioral Design
    * Use tasks, always block
* RTL
    * Explicit clock



Multiplexor: Gate Level

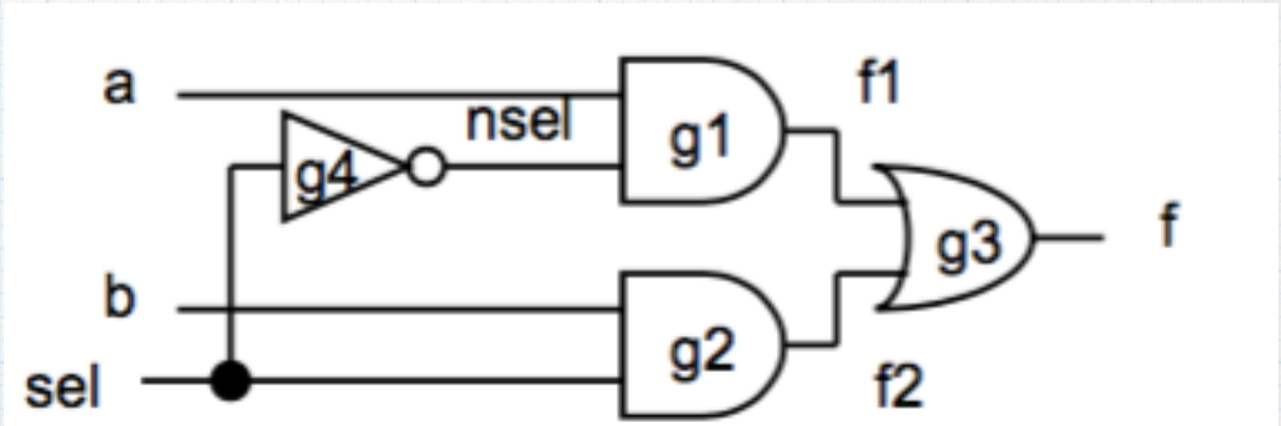
```
module mux(f, a, b, sel);

output f;
input a, b, sel;

      and g1(f1, a, nsel),
            g2(f2, b, sel);


      or g3(f, f1, f2);

      not g4(nsel, sel);


endmodule
```



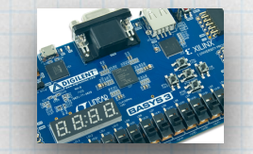


Multiplexor: Behavioral Level

```
module mux(f, a, b, sel);

output f;
input a, b, sel;
reg f;
```

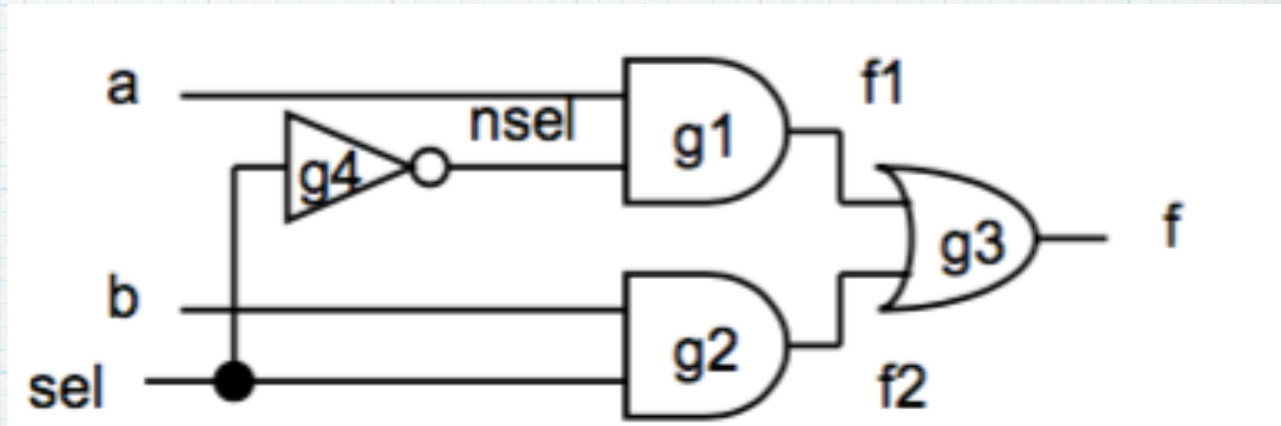


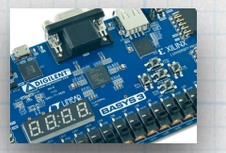

```
always @(a or b or sel)

 if (sel) f = a;
 else f = b;

endmodule
```



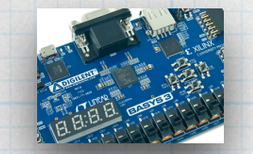
Multiplexor: Continuous Assignment

```
module mux(f, a, b, sel);

output f;
input a, b, sel;

assign f = sel ? a : b;

endmodule
```





Pitfalls

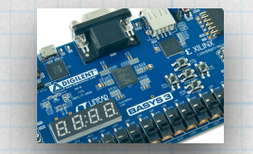


Common Pitfall

* Verilog is not a programming language, but a HDL.

* Don't try to debug a design that doesn't work.

* Don't use Latches or Flip-Flops in combinational circuit.

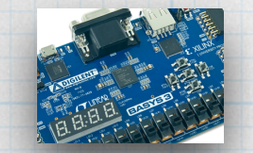


Blocking vs. NonBlocking

```
module shiftReg1(out,d, clock);
output reg [3:0] out;
input d, clock;
always @(posedge clock) begin
        out[0]=d;
        out[1]=out[0];
        out[2]=out[1];
        out[3]=out[2];
end
endmodule
```

* blocking assignments

```
module shiftReg2(out,d, clock);
output reg [3:0] out;
input d, clock;
always @(posedge clock) begin
        out[3]=out[2];
        out[2]=out[1];
        out[1]=out[0];
        out[0]=d;
end
endmodule
```

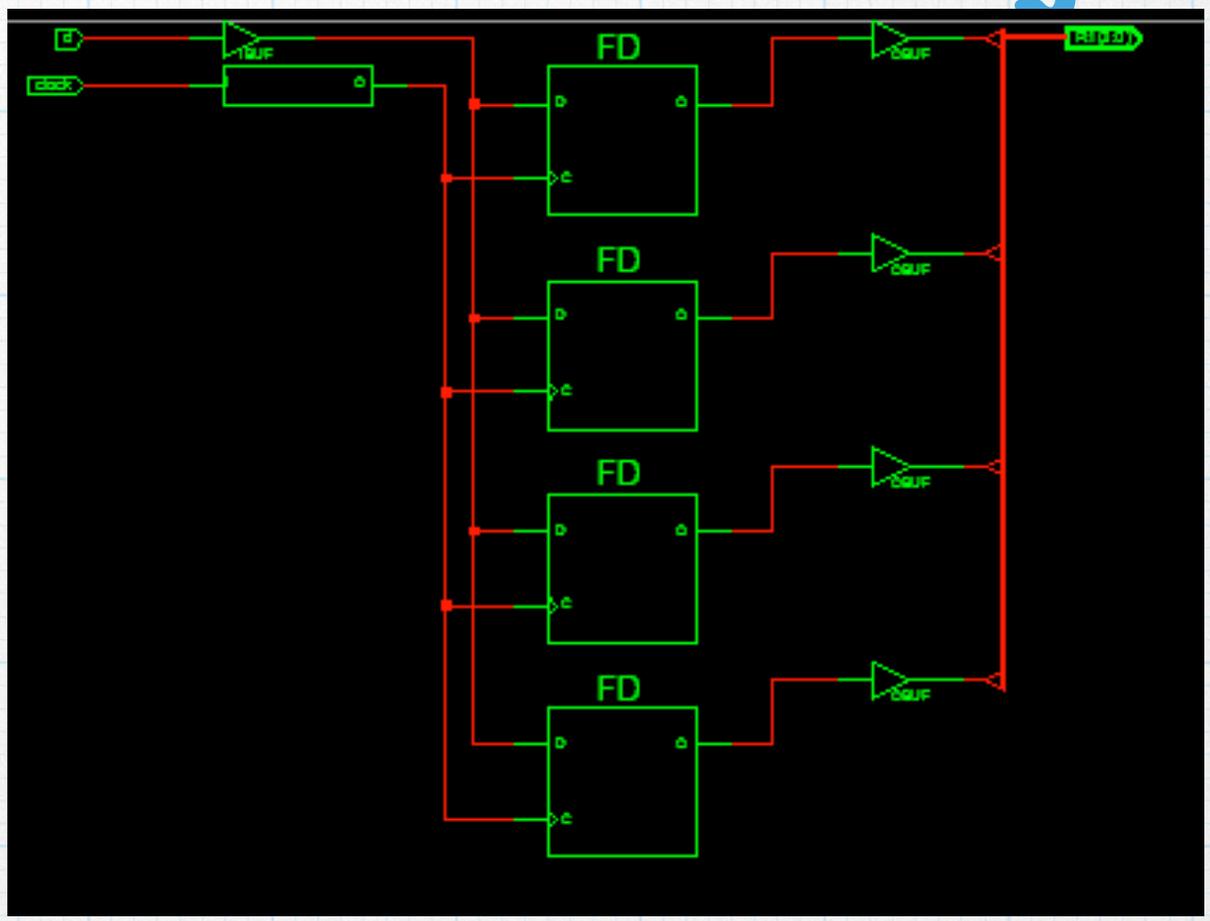


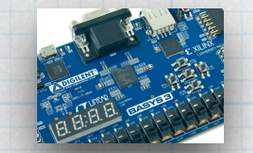
Blocking vs. NonBlocking

```
module shiftReg1(out,d, clock);
output reg [3:0] out;
input d, clock;
always @(posedge clock) begin
        out[0]=d;
        out[1]=out[0];
        out[2]=out[1];
        out[3]=out[2];
end
endmodule
```

* blocking assignments

```
module shiftReg2(out,d, clock);
output reg [3:0] out;
input d, clock;
always @(posedge clock) begin
        out[3]=out[2];
        out[2]=out[1];
        out[1]=out[0];
        out[0]=d;
end
endmodule
```



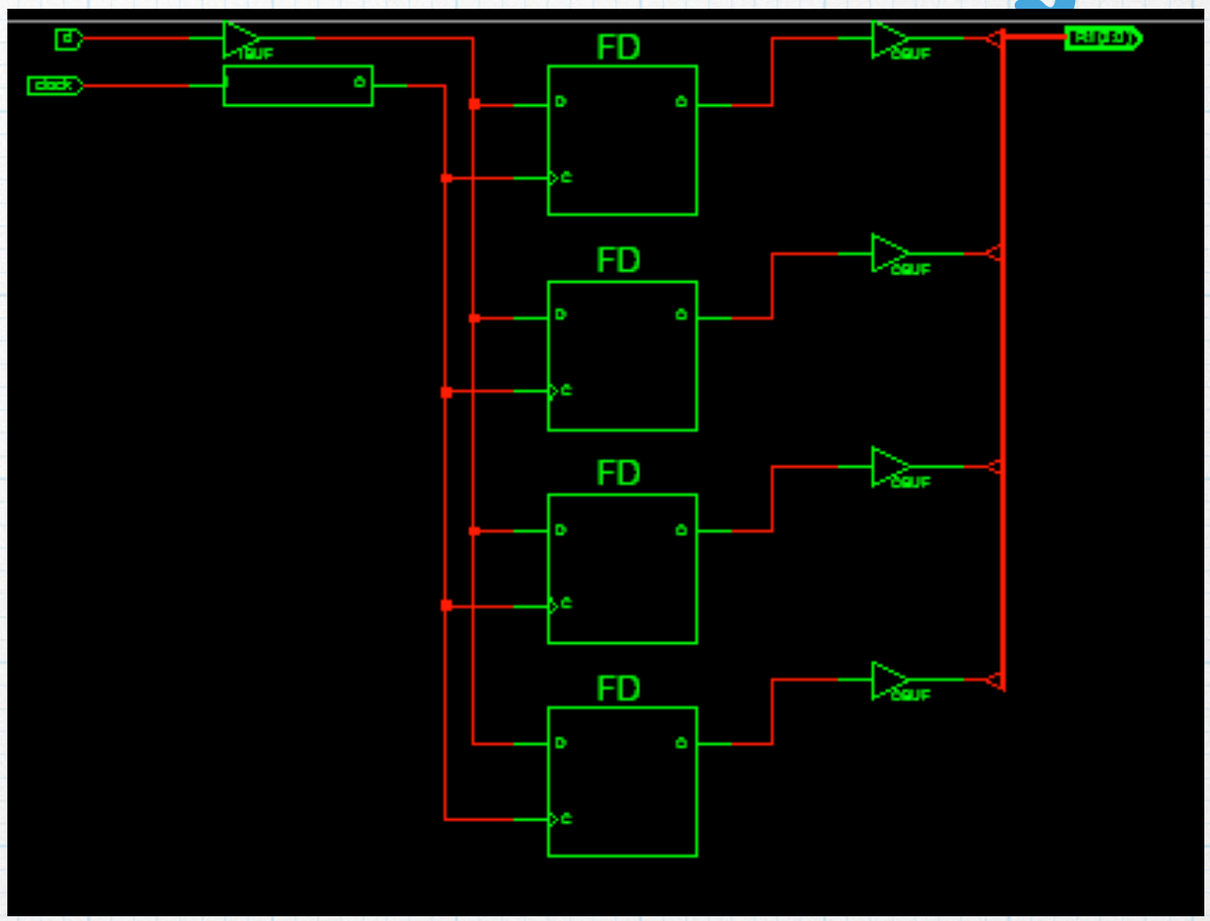


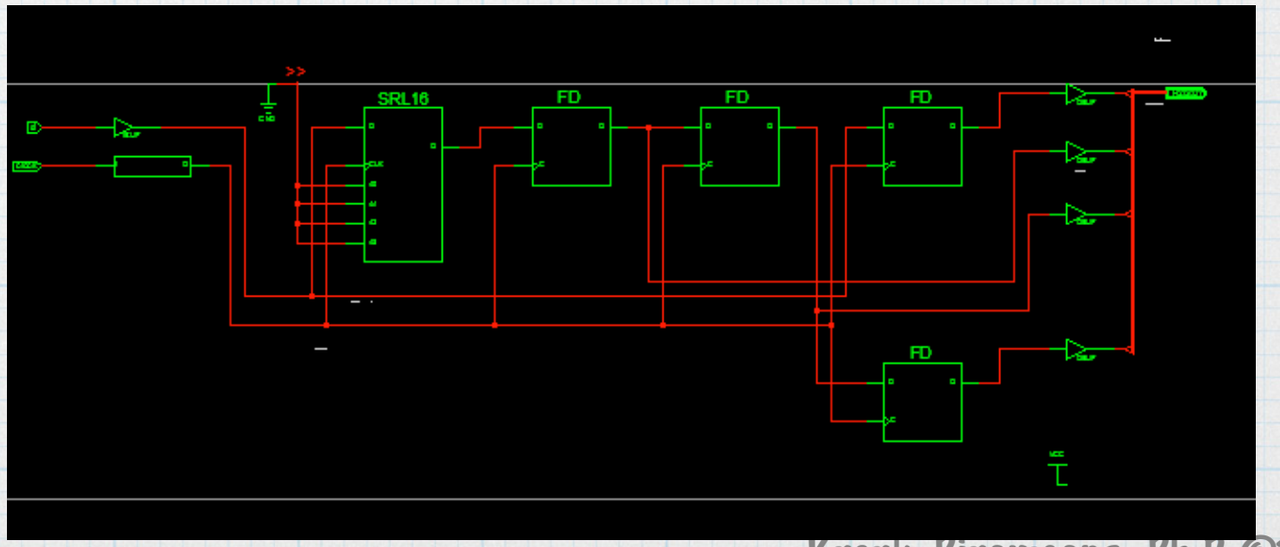
Blocking vs. NonBlocking

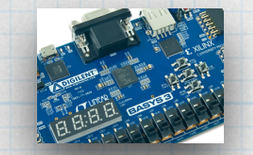
```
module shiftReg1(out,d, clock);
output reg [3:0] out;
input d, clock;
always @(posedge clock) begin
        out[0]=d;
        out[1]=out[0];
        out[2]=out[1];
        out[3]=out[2];
end
endmodule
```

* blocking assignments

```
module shiftReg2(out,d, clock);
output reg [3:0] out;
input d, clock;
always @(posedge clock) begin
        out[3]=out[2];
        out[2]=out[1];
        out[1]=out[0];
        out[0]=d;
end
endmodule
```





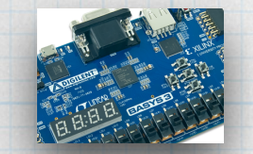


Blocking vs. NonBlocking

* Nonblocking assignments

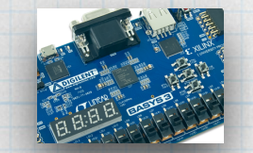
```
module shiftReg3(out,d, clock);
output reg [3:0] out;
input d, clock;

always @(posedge clock) begin
      out[0]<=d;
      out[1]<=out[0];
      out[2]<=out[1];
      out[3]<=out[2];
end

endmodule
```

```
module shiftReg4(out,d, clock);
output reg [3:0] out;
input d, clock;

always @(posedge clock) begin
  out[3]<=out[2];
      out[2]<=out[1];
      out[1]<=out[0];
  out[0]<=d;
end

endmodule
```

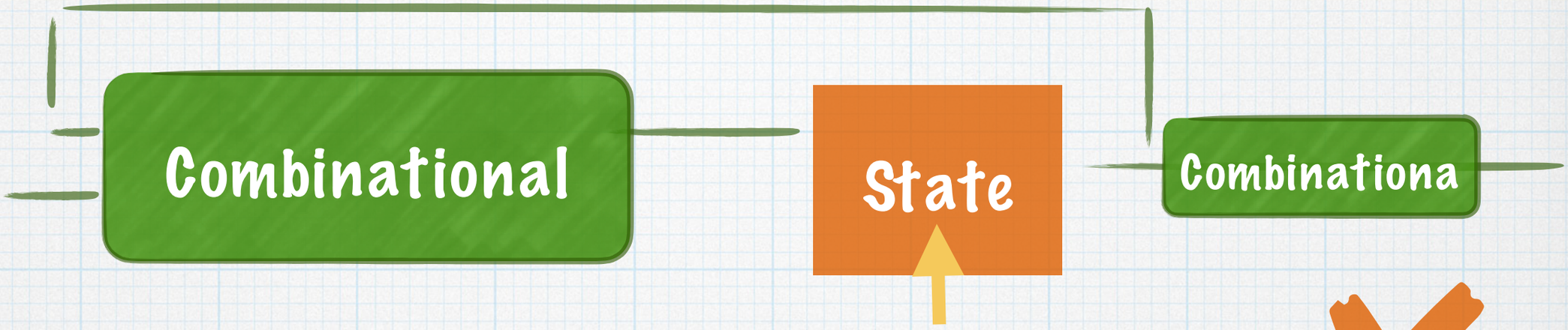


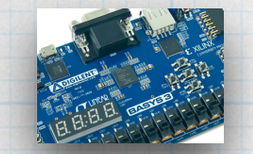
State Machine

* Finite State Machine (FSM)
  * state register
  * combinational logic



Good vs. Bad





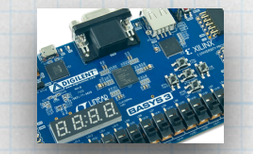
Good vs. Bad

Combinational

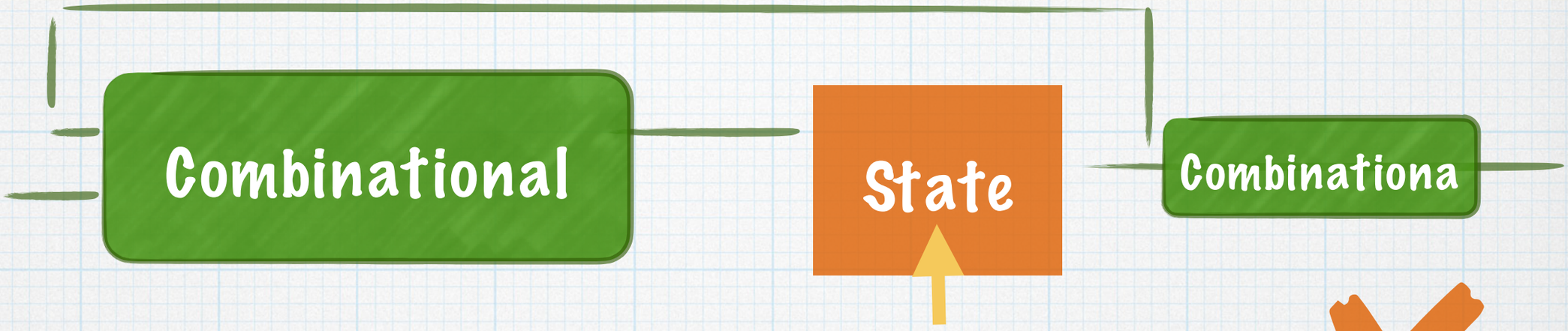
State

Combinationa

```
always @(posedge clock) begin
    ps=ns;
end

always @([input], ps) begin

end
```

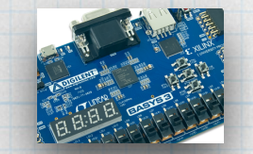


Good vs. Bad



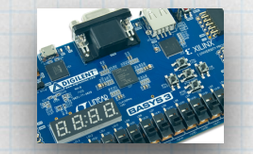

```
always @(posedge clock) begin
    ps=ns;
end

always @([input], ps) begin

end
```

```
always @(posedge clock)
begin
case ...
....
....
...
end
```



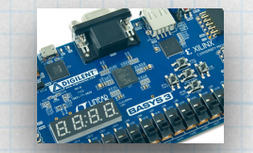
"Think as a circuit designer, not a programmer"

VerilogHDL GuideLine -Krerk Piromsopa, Ph.D.



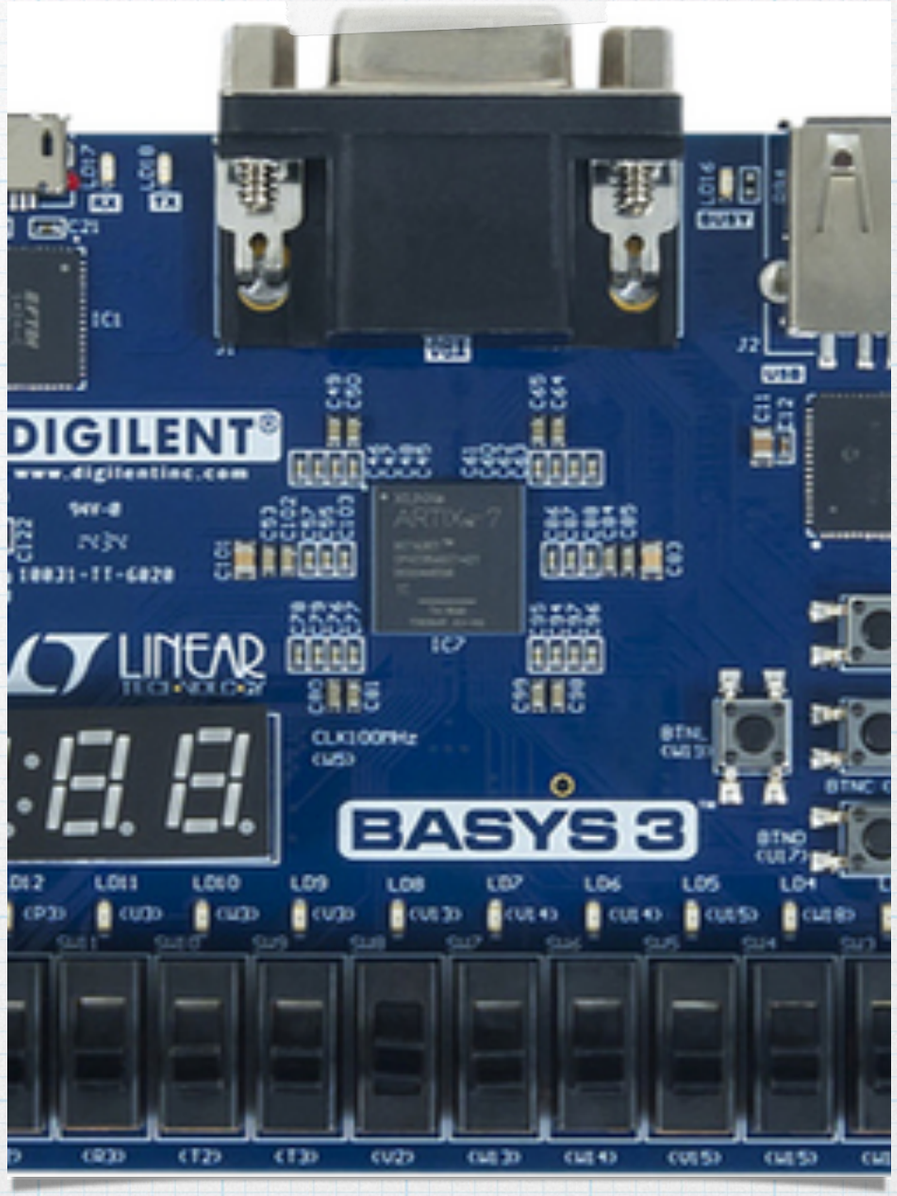
Our tools

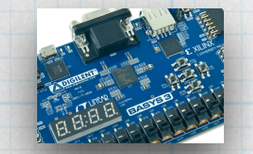
* Xilinx Vivado
* http://www.xilinx.com/
* ~~For simulation, we used~~
  * ~~Simulator: iVerilog~~
  * ~~Waveform Viewer: gtkWave~~



Our Device

* DIGILENT BASYS3
  * Features the Xilinx Artix-7 FPGA: XC7A35T-1CPG236C
  * 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)
  * 1,800 Kbits of fast block RAM
  * Five clock management tiles, each with a phase-locked loop (PLL)
  * Internal clock speeds exceeding 450 MHz
  * On-chip analog-to-digital converter (XADC)
  * VGA, USB HID (mouse, keyboard)





References

* on-line resources:
  * http://www.sutherland-hdl.com/on-line_ref_guide/vlog_ref_top.html
  * http://www.engineering.usu.edu/classes/ece/2530/appendb.pdf
  * http://www.asic-world.com/verilog/vqref.html
  * http://oldeee.see.ed.ac.uk/~gerard/Teach/Verilog/
* Google is your friend.