Instantiating Library Cells

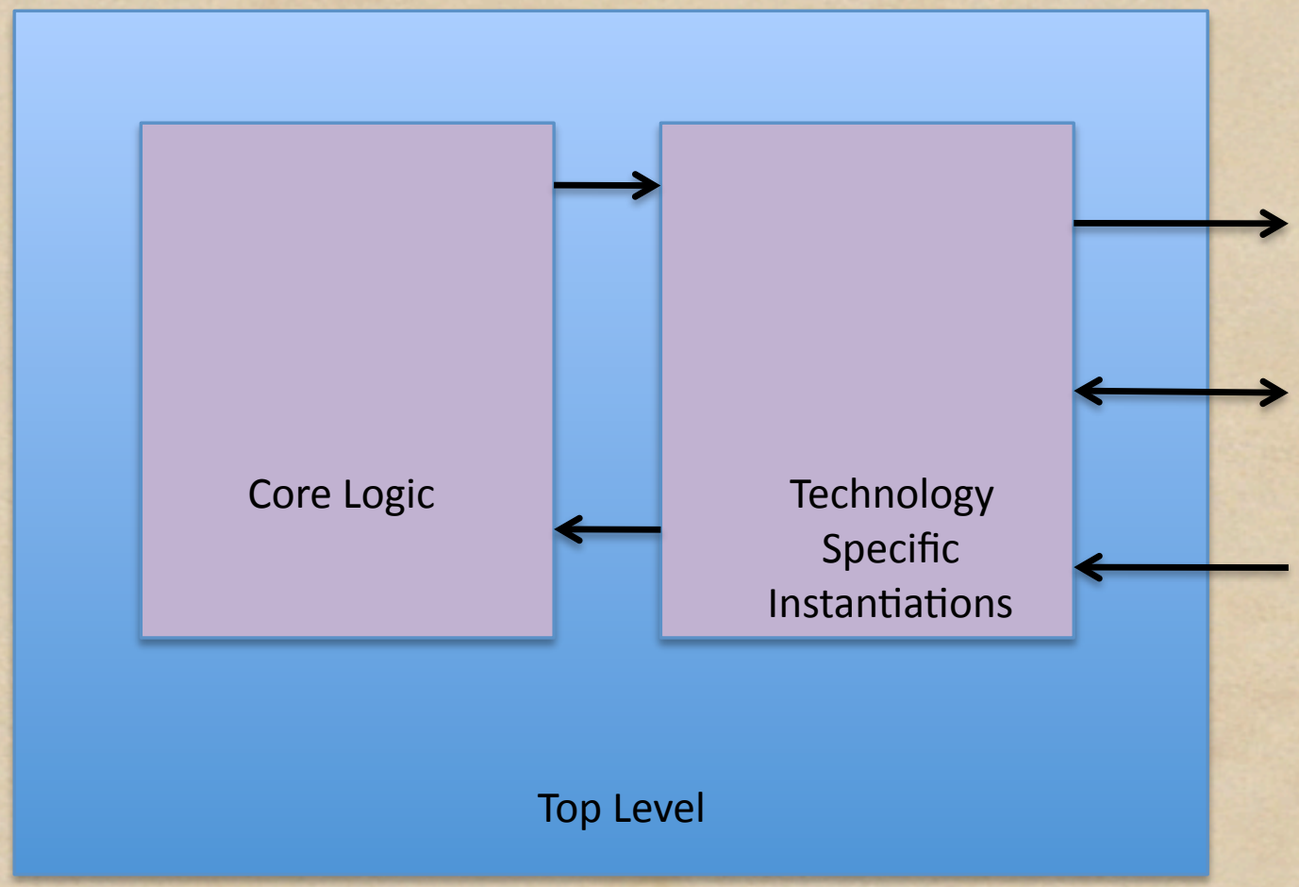
CECS 460

CSULB

John Tramel

Top Level Architecture

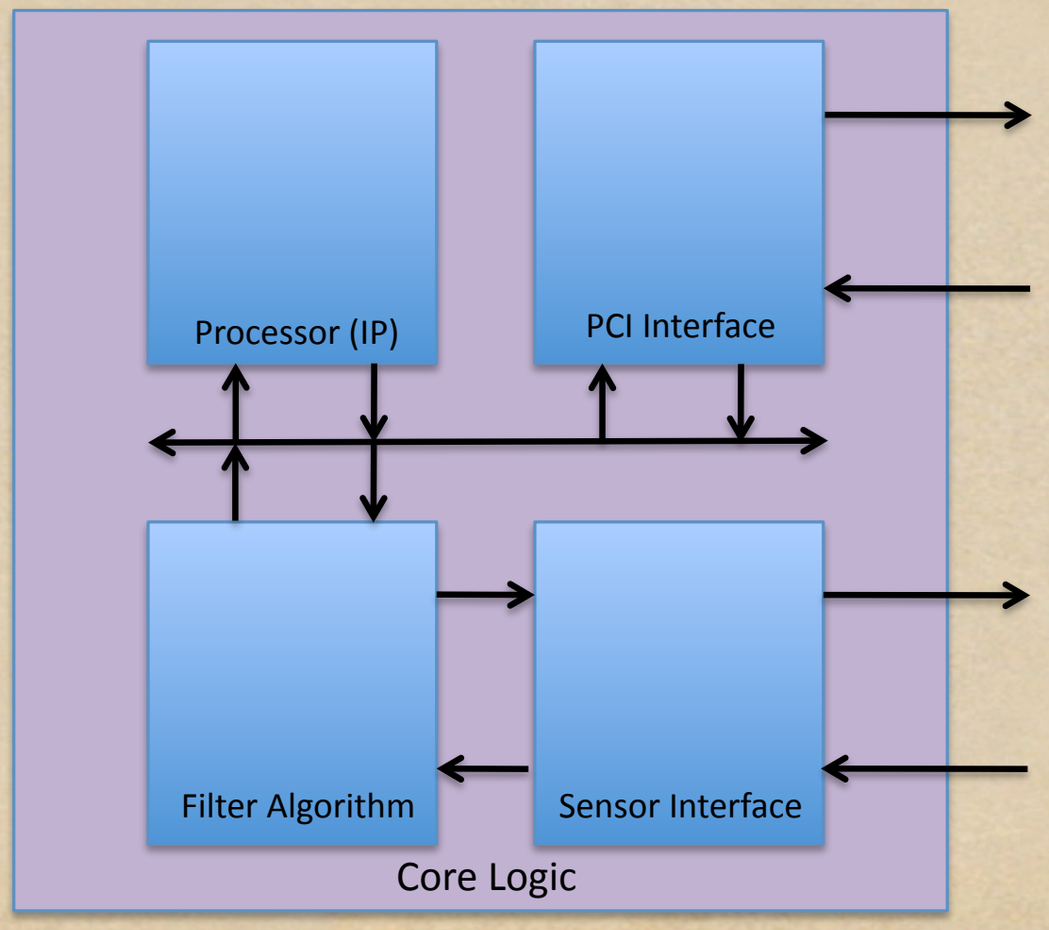
- The typical design is partitioned as follows



- Core Logic contains the design
- Technology Specific Instantiations (TSI) contains all references to the target technology library
- All communication to/from the I/O of the device pass through the TSI

Core Logic Organization

- The core logic is organized by the major functional blocks
- The functional blocks may be blocks designed by the team or IP designed by another organization
- Each block is thoroughly described and defined by the chip specification



Technology Specific Instantiations

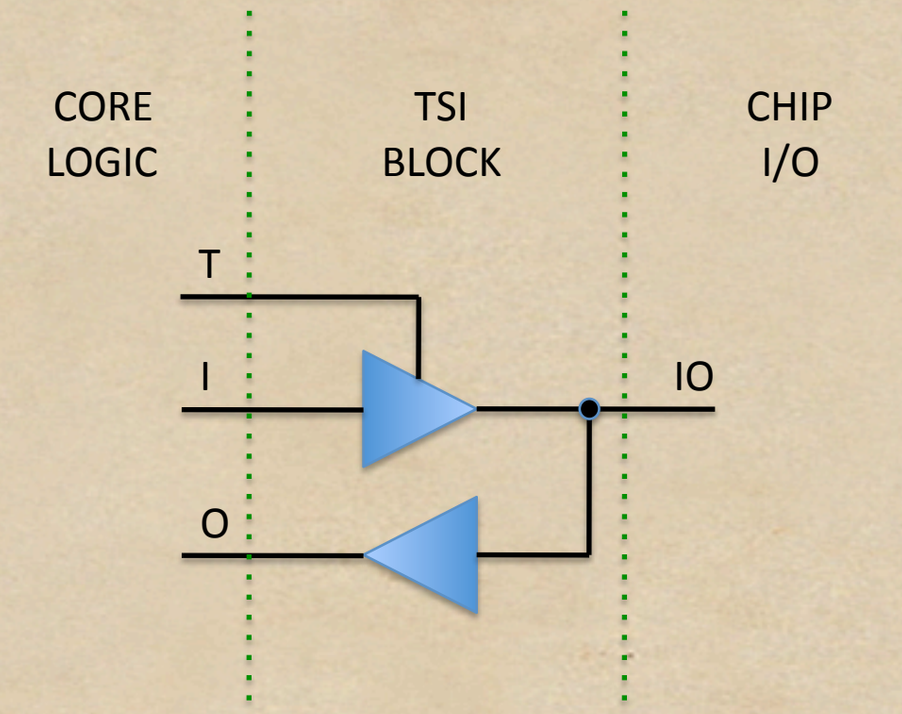
- The I/O cells for a design are one of the primary inclusions of the TSI block
- Each I/O of the chip must have a particularly selected device to meet electrical and timing requirements of the external interface
- The manufacturer's library will define the cells and their capabilities

Spartan-3E I/O Components

IBUFDS	Primitive: Differential Signaling Input Buffer with Selectable I/O Interface
IBUFG	Primitive: Dedicated Input Buffer with Selectable I/O Interface
IBUFGDS	Primitive: Dedicated Differential SIgnaling INput Buffer with Selectable I/O Interface
IDDR2	Primitive: Dual Data Rate Input D Flip-Flop with Optional Data Alignment, Clock Enable and Programmable Synchronous or Asynchronous Set/Reset
ODDR2	Primitive: Dual Data Rate Output D Flip-Flop with Optional Data Alignment, Clock Enable and Programmable Synchronous or Asynchronous Set/Reset
IOBUF	Primitive: Bi-Directional Buffer with Selectable I/O Interface (multiple primitives)
IOBUFDS	Primitive: 3-State Differential Signaling I/O Buffer with Active Low Output Enable
KEEPER	Primitive: KEEPER Symbol
OBUF	Primitive: Single- and Multiple-Output Buffer
OBUFDS	Primitive: Differential Signaling Output Buffer with Selectable I/O Interface
OBUFT	Primitive: 3-State Output Buffer with Active Low Output Enable
OBUFTDS	Primitive: 2-State Output Buffer with Differential Signaling Active-Low Output Enable, Selectable I/O Interface
PULLDOWN	Primitive: Resistor to GND for Input Pads
PULLUP	Primitive: Resistor to VCC for Input Pads, Open-Drain, and 3-State Outputs

Example I/O Component

- IOBUF - Primitive: Bi-Directional Buffer with Selectable I/O Interface



- The I/O on the chip is a bi-directional interface (DQ on memory)
- The core logic must supply the data (I) to go out the IO pin and must receive the data (O) coming in the IO pin
- The T signal is the I/O control on the output buffer

IOBUF Continued

INPUT		BIDIRECTIONAL	OUTPUTS
T	**I**	**IO**	**O**
1	X	Z	X
0	1	1	1
0	0	0	0

- IOBUFs are composites IBUF and OBUFT elements
- Available Attributes
  - LVTTL, LVCMOS15, LVCMOS18, LVCMOS25 and LVCMOS33
    - Require a slew value (FAST or SLOW) and DRIVE value

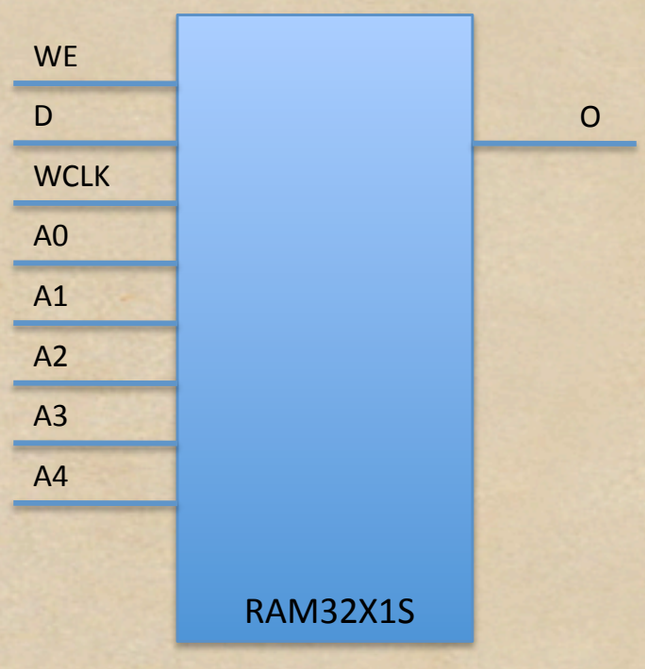
Xilinx Spartan-3E IOSTANDARDs

Single-Ended IOSTANDARD	1.5V	1.8V	2.5V	3.3V
LVTTL	-	-	-	Input/ Output
LVCMOS33	-	-	-	Input/ Output
LVCMOS25	-	-	Input/ Output	Input
LVCMOS33	-	Input/ Output	Input	Input
LVCMOS25	Input/ Output	Input	Input	Input

- Spartan-3E FPGAs support the following single-ended standards: 3.3V low-voltage TTL (LVTTL), Low-voltage CMOS (LVCMOS) at 3.3V, 2.5V, 1.8V, 1.5V, or 1.2V; 3V PCI at 33 MHz
- Spartan-3E FPGAs also support the following differential standards: LVDS, Bus LVDS, mini-LVDS, RSDS ...

Instantiating Xilinx Memory

- The Spartan-3E Library has memory cells that may be instantiated directly into a Verilog (or VHDL) design



Primitive: 32-Deep by 1-Wide Static Synchronous RAM

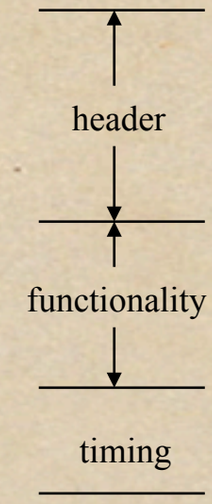
```
// RAM32X1S: 32 X 1 posedge write

RAM32X1S RAM32X1S_inst (
   .O(O),
   .A0(A0),
   .A1(A1),
   .A2(A2),
   .A3(A3),
   .A4(A4),
   .D(D),
   .WCLK(WCLK),
   .WE(WE)
  ;

// Edit the following defparam to
// change initial value of RAM

defparam RAM32x1S_inst.INIT = 32h0;
```

The World Below

- As we have studied computer modeling is based on a series of abstractions. Each level with added responsibility to behave as close to the physical implementation as possible
- When modeling with Verilog this approach holds true. At the top level we model with RTL (Register Transfer Level) representation where the only concept of time is waiting for the active edge of the clock
- Once the design is synthesized to gates there is the need to consider the switching delays of the gates and the output load delays in order to ensure that the design will function once implemented
- The Verilog libraries that implement the gates are very similar to the modules we have designed with - the primary difference is the addition of a third portion of the module - the specify block

The Complete Verilog Module

- The basic building block in Verilog is the module
  - Delimited by keywords *module* and *endmodule*
  - Represents autonomous processes with defined interfaces
  - A Verilog design is composed of one or more modules
- The circuit being modeled and the test bench will be modules
- The structure of a Verilog module




```
module andgate (a, b, y);       // keyword, module name, port list
input      a;
input      b;
output     y;                   // define all ports as input, output or inout

and a1 (y, a, b);               // functionality defined using Verilog primitive

specify …                       // typically library cells only have specify
endspecify

endmodule
```

Specify Block Entries

(a => b) = 1.8;	parallel connection path; one delay for all output transitions
(a -*> b) = 2:3:4;	full connection path; one min:typ:max delay range for all output transitions; b receives the inverted value of a
specparam t1 = 3:4:6, t2 = 2:3:4; (a => y) = (t1,t2);	different path delays for rise, fall transitions
(a *> y1,y2) = (2,3,4,3,4,3);	different path delays for rise, fall transitions
(posedge clk => (qb -: d)) = (2.6, 1.8);	edge-sensitive path delay; timing path is positive edge of clock to qb; qb receives the inverted value of data
if (rst && pst) (posedge clk=>(q +: d))=2;	state-dependent edge sensitive path delay

if (opcode = 3‘b000)
(a,b *> o) = 15;
if (opcode = 3‘b001)
(a,b *> o) = 25;
ifnone
(a,b *>o) = 10;
	state-dependent path delays; an ALU with different delays for certain operations (default delay has no condition)

Sample Specify Block

```
// Conditional Path Delay
module Mod_X(out, a, b, c);
output out;
input a, b, c;
wire d;
// Specify block assigning conditional path delays
specify

// Different path delay dependent on signal c state
if (c) (c => out) = 12;
if (~c) (c => out) = 8;

// Use of two operands
if (a | b) (a => out) = 10;
if (~(a | b)) (a => out) = 12;

// Conditional definition using Concatenation
if ({a,b} == 2'b11) (b => out) = 16;
if ({a,b} != 2'b11) (b => out) = 13;
endspecify

or o1 (d, a, b);
and a1 (out, d, c);

endmodule
```

Library Cell Definition

- Many library cells are exclusively modeled using primitives. Either built-in primitives or user-defined primitives
- The built-in primitives are:
  - and nand or nor xor xnor buf bufif0 bufif1 bot notif0 notif1 pulldown pullup
- When instantiated, the first argument is the output and successive arguments are the inputs

Built-In Primitive Behavior

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

nand	0	1	x	z
0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

nor	0	1	x	z
0	1	0	x	x
1	0	0	0	0
x	x	0	x	x
z	x	0	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

xnor	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

User-Defined Primitive - MUX

```
primitive mux (y, a, b, sel); //COMBINATIONAL UDP
   output y;
   input sel, a, b;
table //table order for inputs matches primitive statement
// a b sel : y
   0 ? 0 : 0;   //select a; don't care about b
   1 ? 0 : 1;   //select a; don't care about b
   ? 0 1 : 0;   //select b; don't care about a
   ? 1 1 : 1;   //select b; don't care about a
endtable

endprimitive
```

User-Defined Primitive - DFF

```
primitive dff (q,d,clk,rst); //SEQUENTIAL UDP
output q;
input clk, rst, d; reg q; //declaring output as reg defines
                          //sequential device with an internal
                          //storage state initial q = 0;
                          //powers up in reset state
table
//d clk rst:state:q
  ? ? 0 : ? :0;     //low true reset
  0 R 1 : ? :0;     //clock in a 0
  1 R 1 : ? :1:     //clock in a 1
  ? N 1 : ? :-;     //ignore negedge of clk
  ? 1 : ? :-;       //ignore all edges on d
  ? ? P : ? :-;     //ignore posedge of rst
  0 (0X) 1 : 0 :-; //reduce pessimism
  1 (0X) 1 : 1 :-; //reduce pessimism endtable endprimitive
```