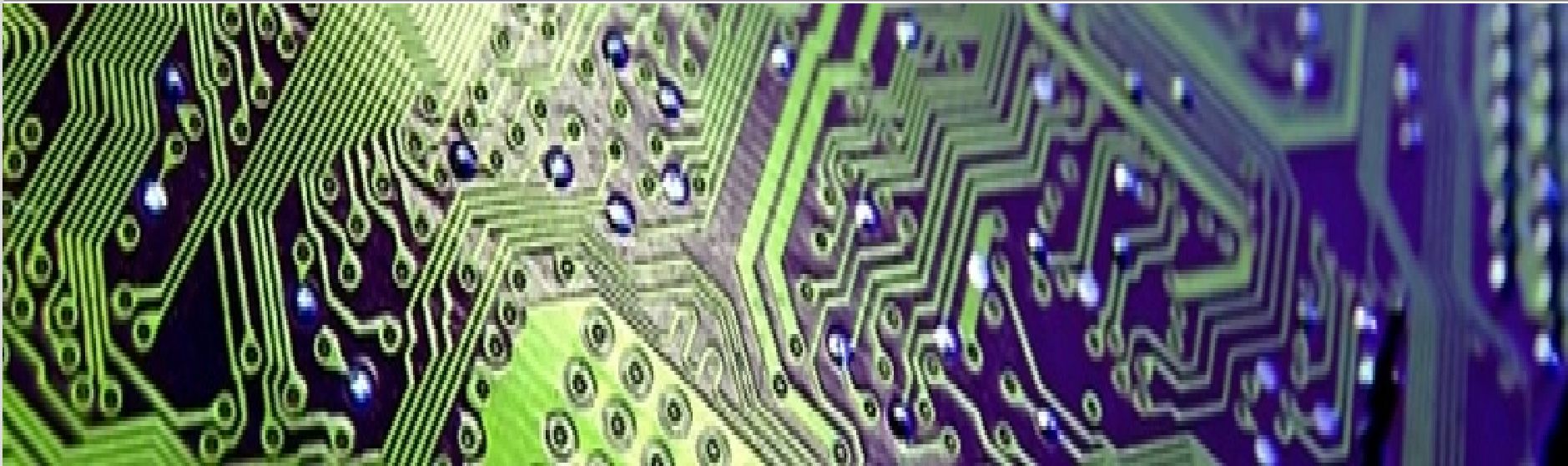


FPGA Practice

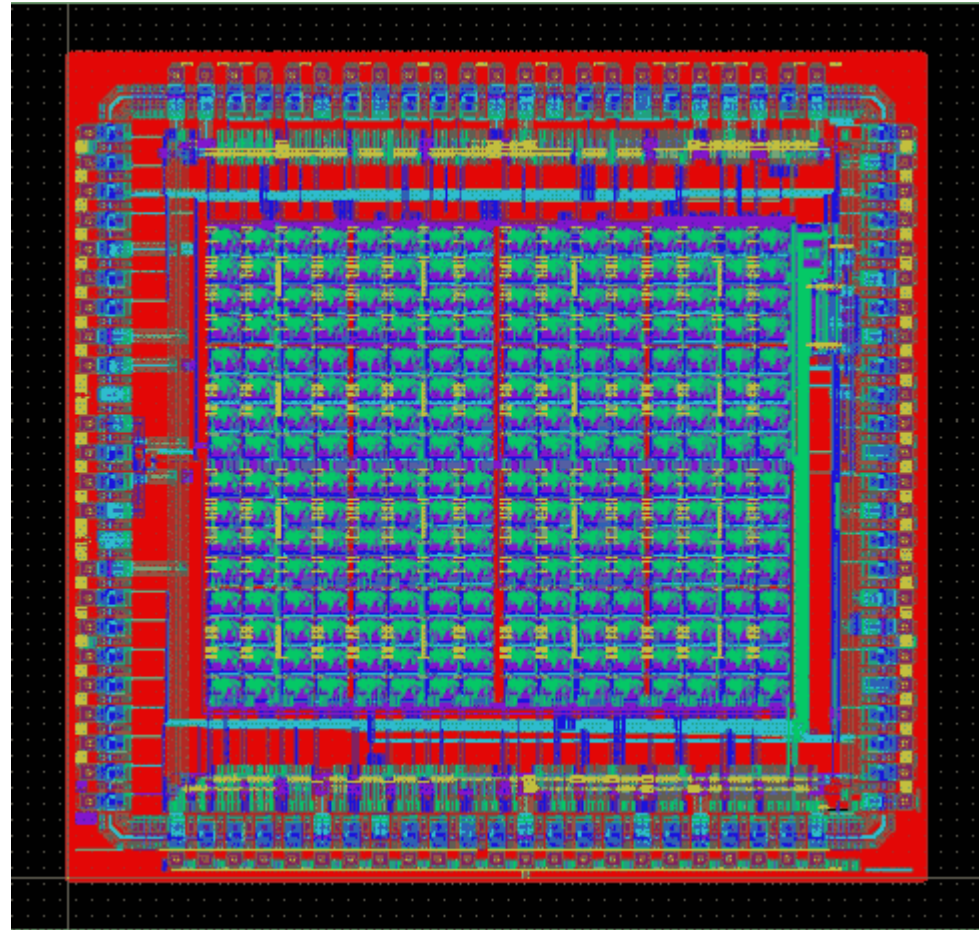
Slides: Dennis Gnad, Arunkumar Vijayan

INSTITUTE OF COMPUTER ENGINEERING (ITEC) – CHAIR FOR DEPENDABLE NANO COMPUTING (CDNC)

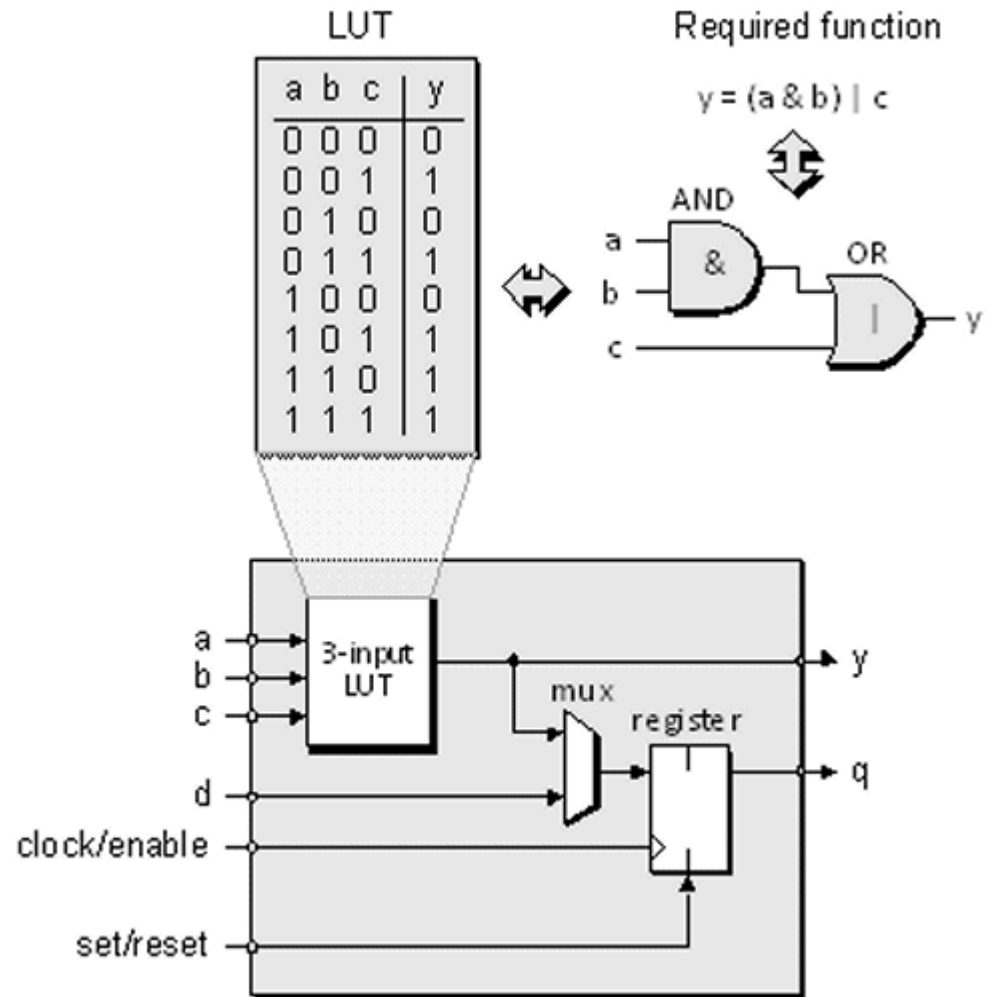
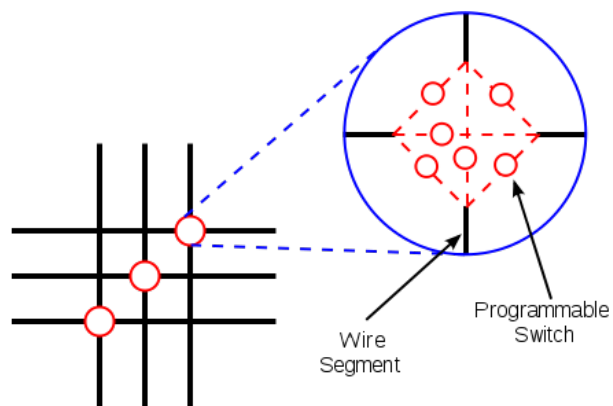
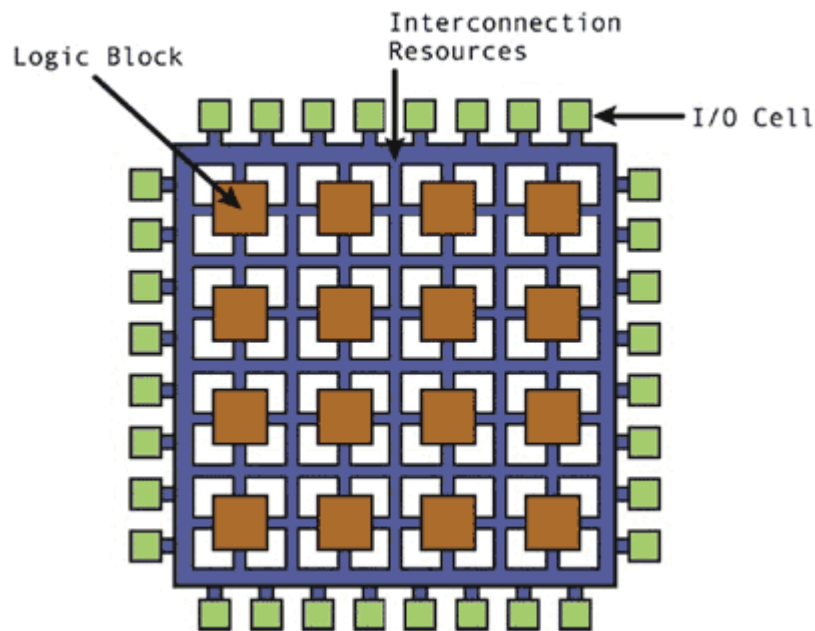


Field-Programmable Gate Array (FPGA)

- **What is an FPGA?**
 - „Programmable Hardware“
 - Array of Programmable Logic Blocks
 - Reconfigurable Interconnects
- **What can we do with it?**
 - Build any combinatorial logic function or state machine using basic logic gates and memory elements.
 - Build complex digital designs on silicon.
- **How is it different from programming an Arduino or a Raspberry Pi?**
 - Describing dedicated logic that exists and computes in parallel, instead of threads in a CPU
- **Or a GPU?**
 - Similar differences to CPU programming
 - OpenCL: Still hard to reach efficient



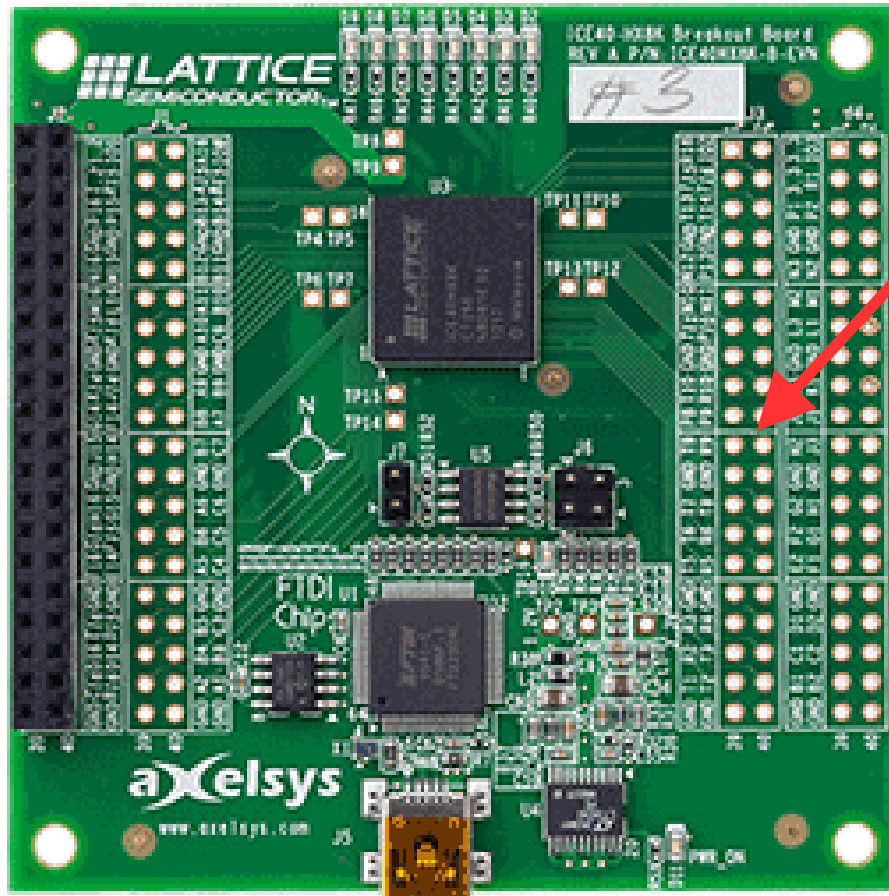
FPGA Architecture



What is a development board?

- Designed to make the development process faster
- Has typical digital inputs/outputs

FPGA



Programming FPGAs

- Design description
 - High-level language like C
 - Needs High-level Synthesis (HLS) support
 - Hardware Description Languages
 - VHDL, Verilog, BlueSpec (high-level)
 - OpenCL
 - Abstracts out hardware-specific details.
- Python (PYNQ)
 - Interfacing and design through Python and kernel libraries

What is HDL?

- HDL is a behavioral description of design in a text format
 - VHDL and Verilog
 - Have similar features and same popularity
 - Could be mixed
- How is HDL code implemented on FPGAs?
 - FPGA vendors provide different tools for this purpose
 - Altera Quartus and Xilinx ISE
 - These tools perform synthesis, place & routing, and timing analysis to map your design to FPGA logic

Verilog Basics

- Comments are similar to C++
 - `//` for single line
 - `/* */` for multi line
- Case-sensitive
- Constants:
 - Integer `223`
 - Hex `8'hAA`
 - Binary `4'b101010`
- Important data types
 - **wire** represents structural connections between components
 - **reg** stores the last value assigned to them until another assignment
 - integer: signed 32 bits reg
 - Declaration: `reg [4:0] counter;` `reg [9:0] load;`
 - Usage: `counter <= load[9:5];` `counter[3] <= load[3]`

Verilog Basics

- **Procedural Block**


- Initial Block

- Used in test benches
 - Not synthesizable
 - Executes only once

- Always Block

- Can describe combinational and sequential logic
 - Has sensitivity list of variables
 - Executes at any **event** in the **sensitivity list**

Sensitivity list



```
always@ (a or b)
Begin
    im1 = a & ~b
    im2 = ~a & b
End
```

- **Assign Statement**

- Describes combinational logic
 - Cannot be used inside a procedural block
 - Do **assign** only to type **wire** (constant driving)
 - Eg: *assign a = b & c*

Example 1: Mux

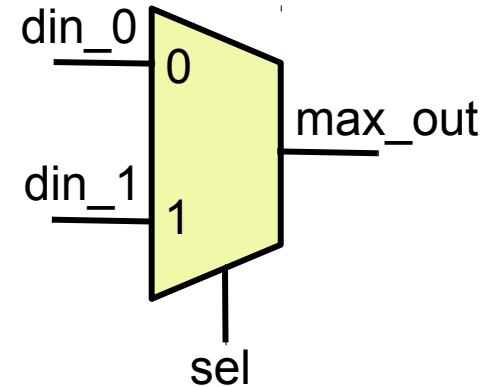
```
module mux_using_case( din_0, din_1, sel, mux_out);  
input din_0, din_1, sel ;  
output mux_out;  
reg mux_out;
```

Port Declaration

```
always @ (sel or din_0 or din_1) begin  
  case(sel)  
    1'b0 : mux_out = din_0;  
    1'b1 : mux_out = din_1;  
  endcase  
end  
endmodule
```

**Behavioral
Description**

```
if (sel == 1'b0) begin  
  mux_out = din_0;  
end else begin  
  mux_out = din_1;  
end
```



Example 2: 8-bit Counter

```
module up_counter (out, enable, clk, reset);
```

```
    output [7:0] out;
```

```
    input enable, clk, reset;
```

```
    reg [7:0] out;
```

Port Declaration

```
    always @(posedge clk)
```

```
        if (reset) begin
```

```
            out <= 8'b0 ;
```

```
        end else if (enable) begin
```

```
            out <= out + 1;
```

```
        end
```

```
    endmodule
```

**Behavioral
Description**

Blocking vs. Non-Blocking Statement

- A verilog statement has two parts:
 - **Evaluation** (evaluating the Right Hand Side (RHS))
 - **Assignment** (assigning the RHS to LHS)
 - **a = b OR c**
 - Evaluation of <b OR c>
 - Assignment of <b OR c> to a

```
always@( a or b or c)
Begin
    P <= a | b | c  evaluates RHS and waits
    Q <= a ^ b      evaluates RHS and waits
    R <= b & c      evaluates RHS and waits
End               Assign P,Q,R based on eval
```

Non-blocking Assignment

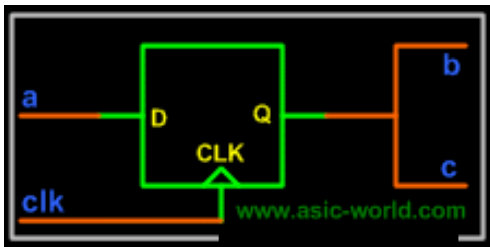
```
always@( a or b or c)
Begin
    P = a | b | c  evaluates RHS and assigns P
    Q = a ^ b      evaluates RHS and assigns Q
    R = b & c      evaluates RHS and assigns R
End
```

Blocking Assignment

Blocking vs. Non-Blocking Statement

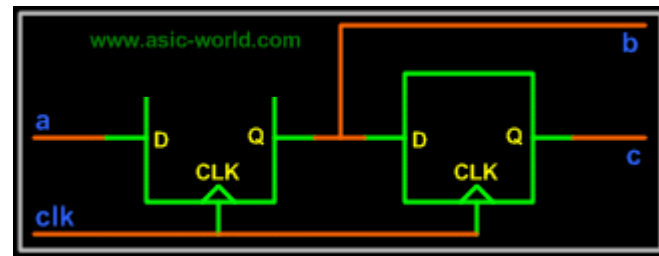
```
module blocking (clk,a,c);  
input clk, a;  
output c;  
wire clk,a ;  
reg c, b;
```

```
always @ (posedge clk )  
begin  
    b = a;  
    c = b;  
end  
endmodule
```



```
module nonblocking (clk,a,c);  
input clk, a;  
output c;  
wire clk,a ;  
reg c, b;
```

```
always @ (posedge clk )  
begin  
    b <= a;  
    c <= b;  
end  
endmodule
```



Verilog Coding Guidelines

Guideline #1: **Sequential logic:** use nonblocking (<=).

Guideline #2: **Latches:** use nonblocking (<=)

Guideline #3: **Combinational logic** with an always block: use blocking (=)

Guideline #4: When modeling both **sequential and combinational logic** within the same always block, use nonblocking assignments (<=).

Guideline #5: **Do not mix** blocking and nonblocking assignments in the same always block.

Guideline #6: Do not make assignments to the same variable from more than one always block (**Multiple Driving**).

(Only for Simulation: **Guideline #7:** Use \$strobe to display values that have been assigned using nonblocking assignments.)

Guideline #8: Do not make assignments using **#0 delays**.

Frequently Encountered Problems and FAQ

- Using 'posedge' on something else than a synchronous clock:
 - This is usually bad practice and will lead to timing issues
- Using both 'negedge' and 'posedge'
 - That is a more advanced topic, not required in these examples
- Don't make too long combinational paths
 - Too long operations on the same 'reg' with blocking assignments (within a single clock cycle)
- Introducing latches
 - Latches can be introduced when any signal needs to be saved in combinational logic (e.g. incomplete case-statement)

Have fun with FPGAs!

Thank You!