Zybo Z7

- https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-getting-started-with-zynq/start
- https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/start?_ga=2.250851654.1799722357.1595866663-1554225925.1573680627&utm_source=digilent&utm_medium=email&utm_campaign=getting+started&utm_content=Zybo+Z7
- https://reference.digilentinc.com/vivado/installing-vivado/start
- Make sure to install the cable drivers on installation!

Verilog HDL

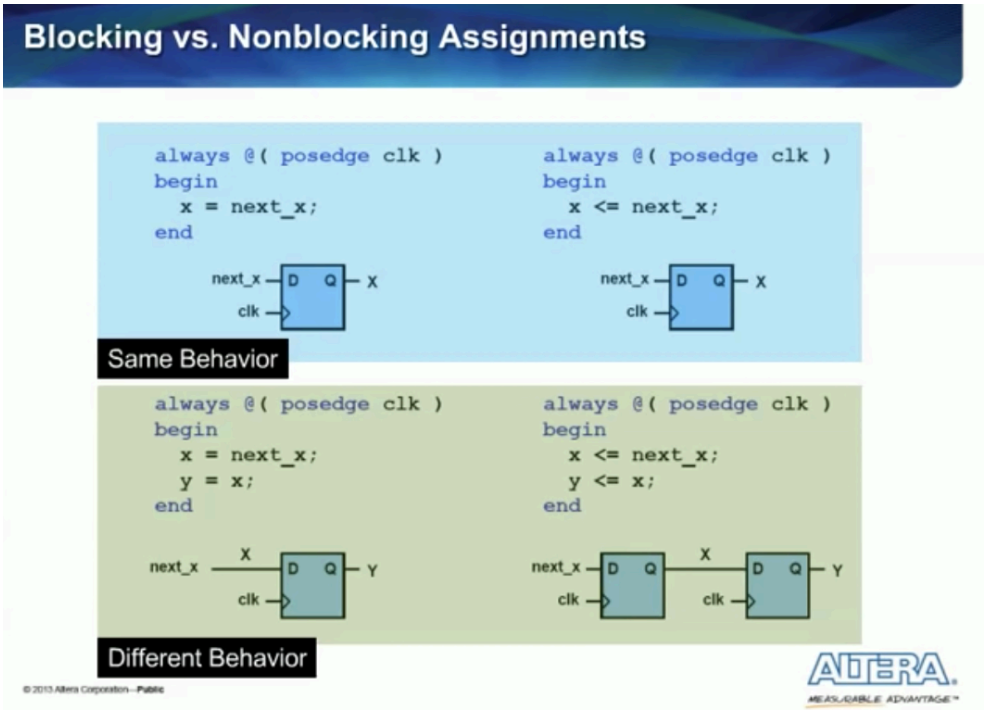
- https://www.youtube.com/watch?v=PJGvZSlsLKs&ab_channel=IntelFPGA
- Simulation and Synthesis
- Simulate at the Behavioral level (Algorithms), RTL Level (Transfer of data between registers, explicit clock signals, Synthesizable), Gate level (Primitive logic gates with AND OR NOR etc., and definite logical values 0, 1, X, Z)
    - Usually gate level code is made by the synthesis tool and not by the user
- Net data type is a physical connection between structures like MUX, Adders
    - Wire type represents a node or connection
    - Tri is a tri state node
    - Supply0 is logic 0
    - Supply1 is logic 1
- Variable data type is for temporary storage
    - Reg type is an unsigned variable of any size
    - Reg signed use signed
    - Integer is a signed 32 bit var
    - Need to be assigned in a procedure, task, or function
    - Cannot be connected to outputs or inputs
- Bus declaration
    - <data_type> [MSB:LSB] name;
- Module Instantiation
    - <component_name> <instance_name> (port list);
    - Component is the lower level module
- Parameter name = 48093.398390 value assigned to a symbolic name
    - Localparam name = 9.1 cannot be changed at compile time
- Numbers
    - [-][Size]'[optional s for signed][d, h, b, o][01000 or some number]
        - X is unknown
        - Z is high impedance
- Operations +, -, *, /, %, ** for exponent

- Bitwise
    - ~ invert, & AND, | OR, ^ XOR, ~^ XNOR
    - Size of largest input
    - A & B, A ~^ B
- Reduction Operator
    - Gives one result by operating on all the bits of an input
    - ~& NAND, ~| NOR, same as bitwise
    - var = 0001
        - &var -> 0
        - |var -> 1
- Relational Operator
    - >, <, >=, <=
    - Equality operators
        - ==, !=
        - ===, !== Case equality and case inequality (Supports all values including X and Z to use them as part of the operation)
- Logic Ops
    - !, &&, ||
- Shifts
    - << SLL, >> SRL, <<< SLA, >>> SRA
        - Arithmetic adds 1s when shifting to the right
- Misc Ops
    - ? operator like in Java
    - {A, B} used to concatenate A and B
- Continuous Assignments
    - wire[4:0] name = A + B;
    - This is ALWAYS being assigned since the wire is a physical connection within the FPGA
- Procedural Assignment Block
    - Initial block is used to initialize behavioral statements
        - Starts at time 0 and does not execute again
    - Always block is always running
        - Starts at time 0 and keeps executing forever
    - Many initial and always can be present in the code

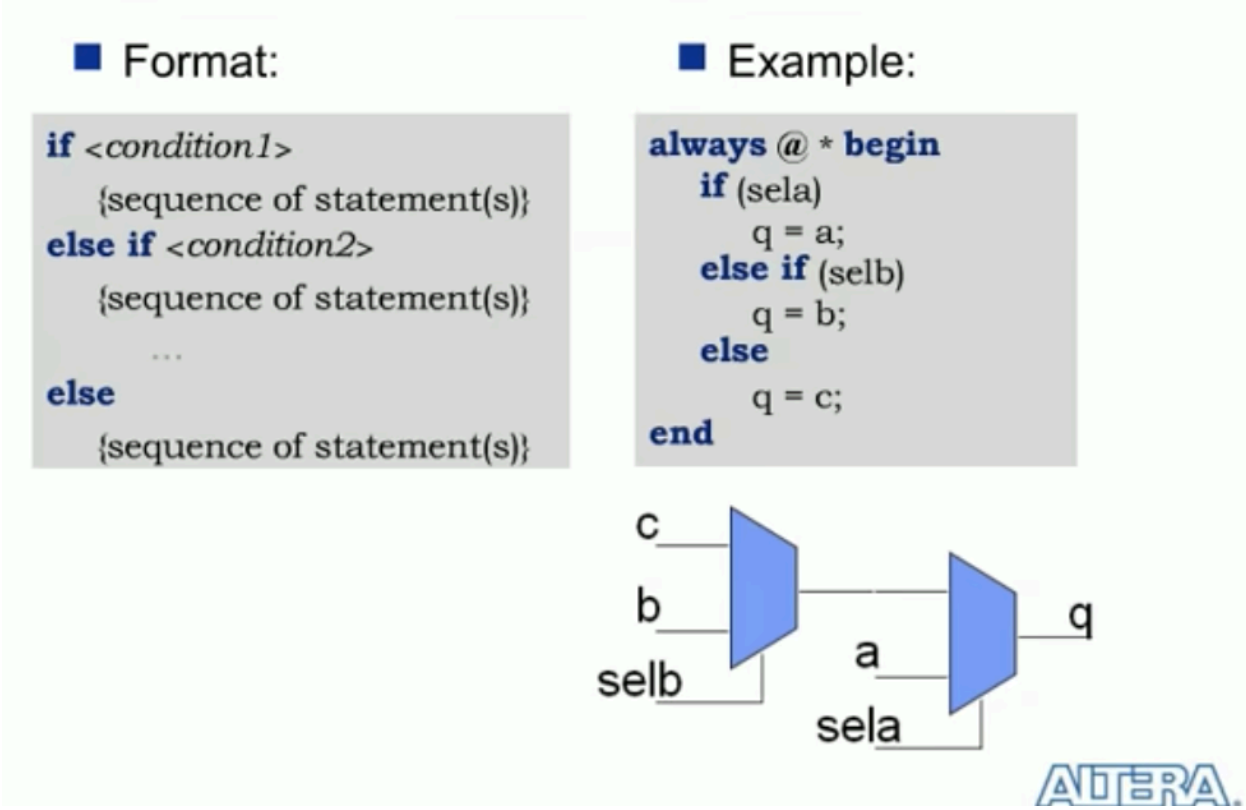
-

```
module clk_gen
    #(parameter period = 50)
(
    output reg clk
);

    initial clk = 1'b0;

    always
        #(period/2) clk = ~clk;

    initial #100  $finish;

endmodule
```

Time	Statement(s) Executed
0	clk = 1'b0;
25	clk = 1'b1;
50	clk = 1'b0;
75	clk = 1'b1;
100	$finish;

    - Toggles the clock every 25 time units and ends after 100 time units
- Blocking vs Non-Blocking Assignment
    - Blocking (Second line cannot be executed until the first delay is done)
        - A = #5 b;
        - C = #10 a;
    - Non-Blocking (Scheduled and second line is executed 10 clock cycles after time 0 rather than 15)
        - A <= #5 b;
        - C <= #10 a;
    - 

- Sensitivity List
    - always @ (posedge clk, negedge clear)

        - This way the code in the always block will only execute when one of the inputs in the sensitivity block changes
    - * used for all signals
- Loops and Such
    - 

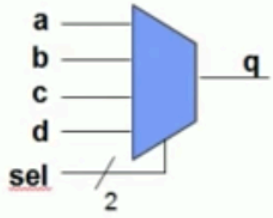
case Statement

- Format:

```
case {expression}
    <condition1> :
        {sequence of statements}
    <condition2> :
        {sequence of statements}
    …
    default : -- (optional)
        {sequence of statements}
endcase
```

- Example:

```
always @ * begin
    case (sel)
        2'b00 : q = a;
        2'b01 : q = b;
        2'b10 : q = c;
        default : q = d;
    endcase
end
```



ALTERA
MEASURABLE ADVANTAGE™

Forever and Repeat Loops

- **forever** loop - executes continually

```
initial begin
    clk = 0;
    forever  #25  clk = ~clk;
end
```

Clock with period of 50 time units

Not synthesizable!

- **repeat** loop - executes a fixed number of times

```
if (rotate == 1)
    repeat (8) begin
        tmp = data[15];
        data = {data << 1, tmp};
    end
```

Repeats a rotate operation 8 times

ALTERA. MEASURABLE ADVANTAGE™

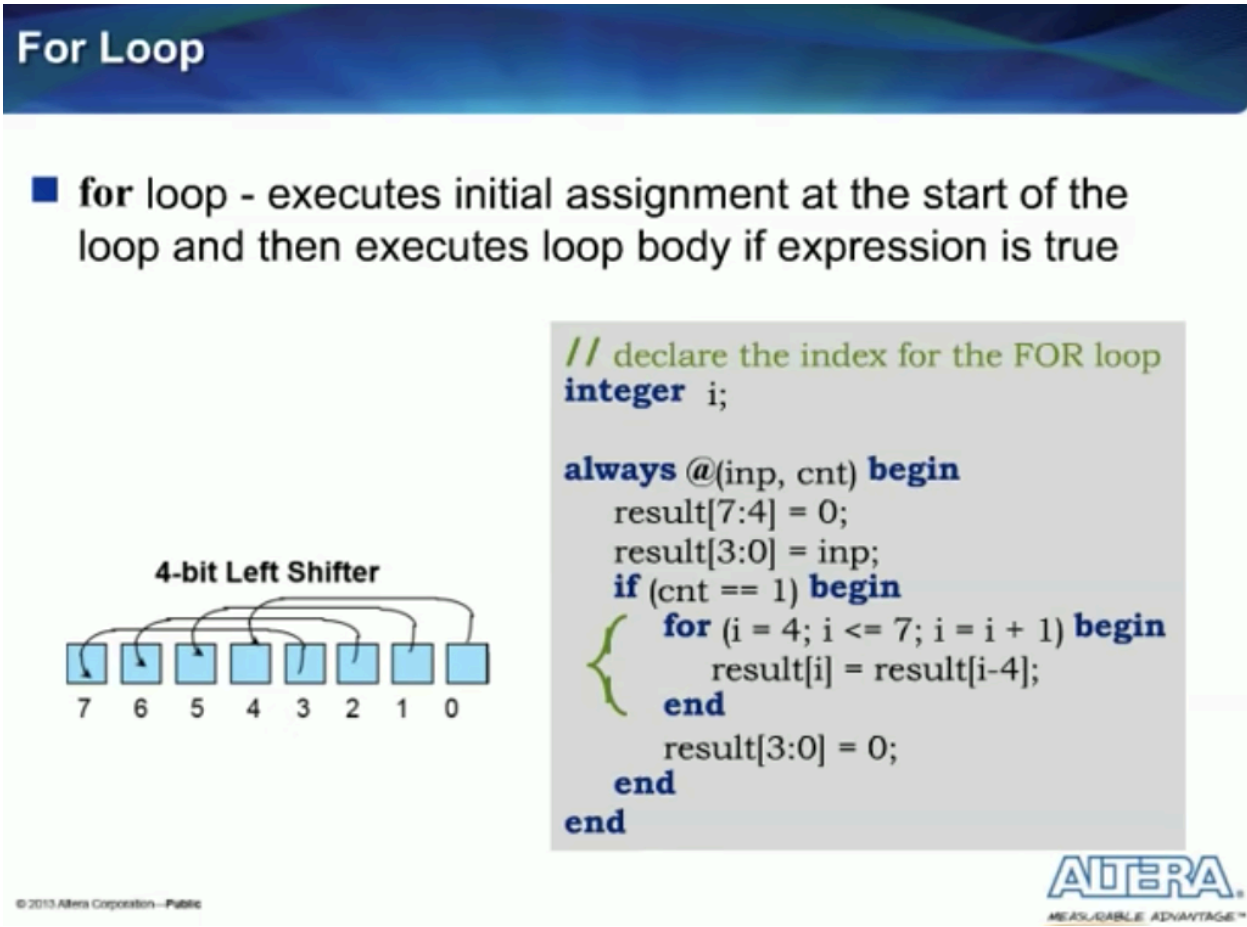
While Loop

- **while** loop - executes if expression is true

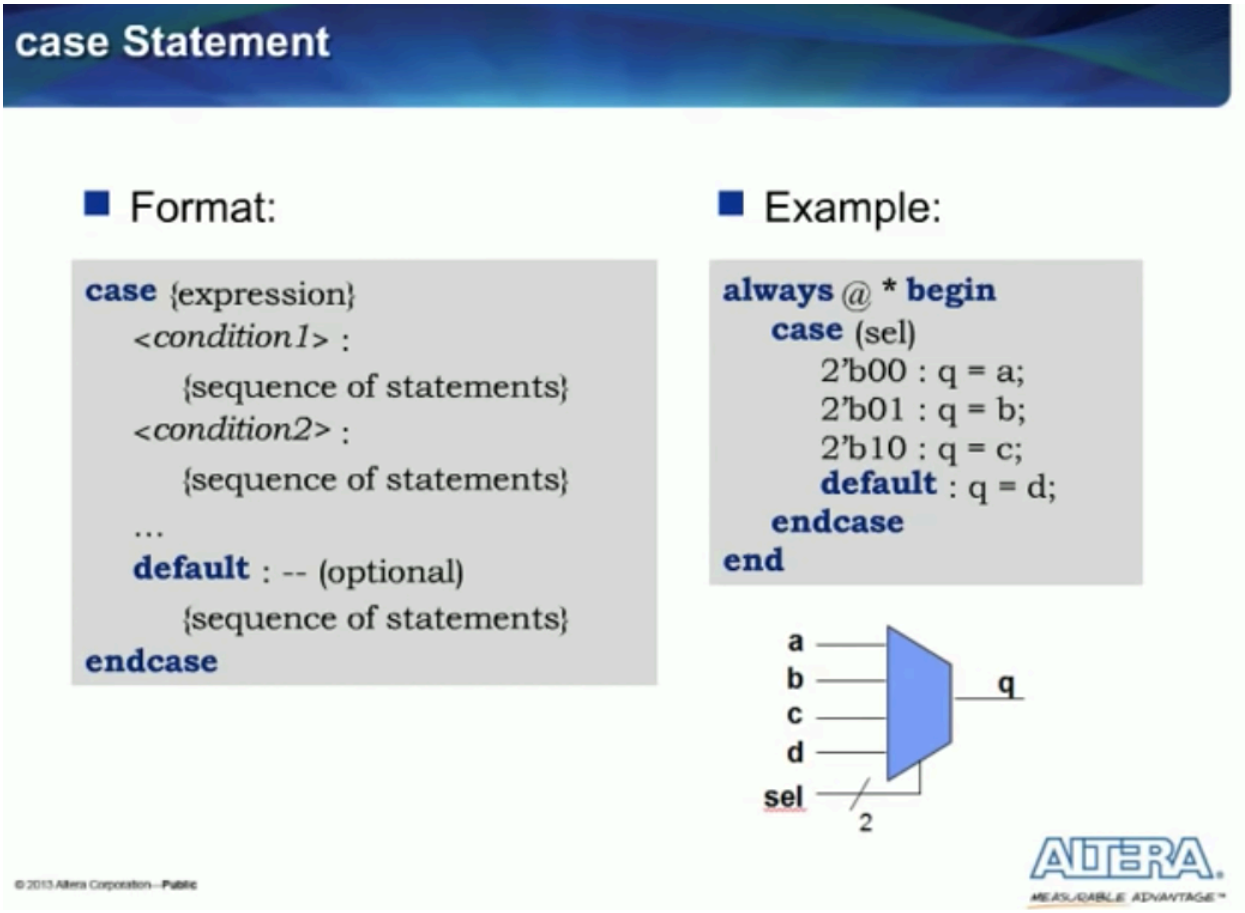
```
initial begin
    count = 0;
    while (count < 101) begin
        $display ("Count = %d", count);
        count = count + 1;
    end
end
```

Counts from 0 to 100
Exits loop at count 101

Not synthesizable!

- 

    - Cannot alter the count variable inside the for loop
- Functions and Tasks
    - Assigned in a module
    - Functions return a value based on the input and combinatorial logic
        - No delays
        - Must have one input arg
        - Must return one arg
    - Tasks are just called and do not produce a value
        - Zero or more arguments

- 

Tips

- Modules always need an output (fixes place 30-494 error)
- DRC NSTD-1 Unspecified I/O standard error: