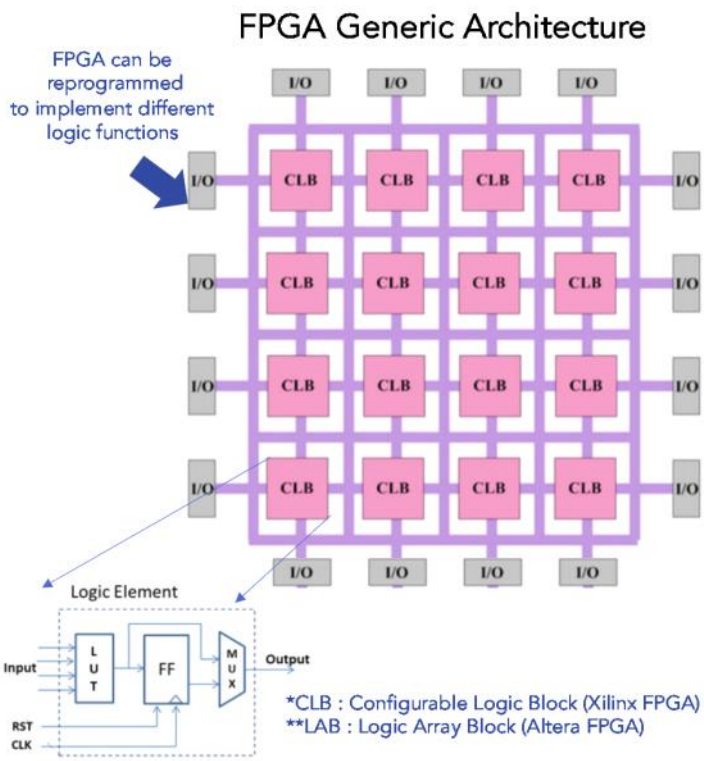
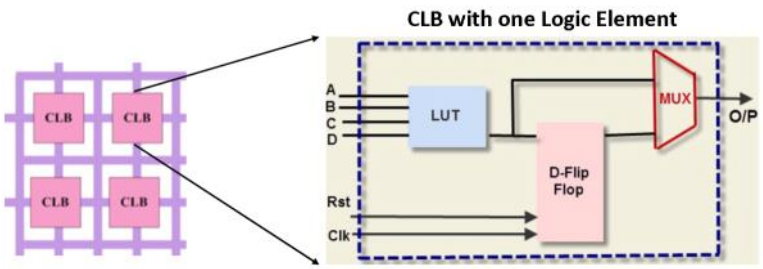
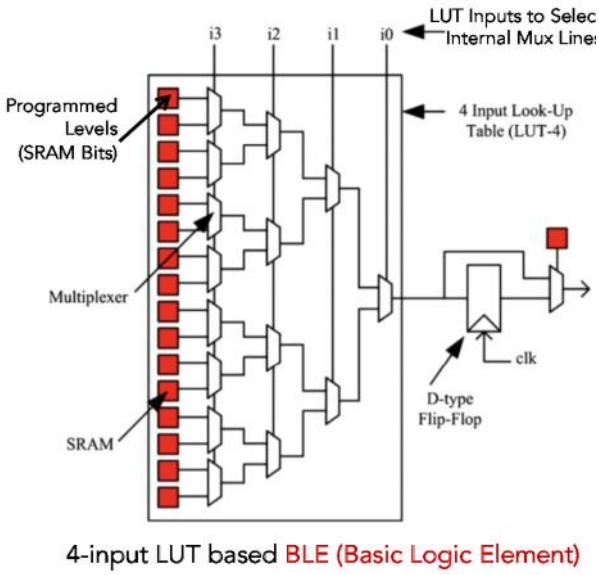
FPGA, Structure, Configuration

Tuesday, March 29, 2022 5:02 PM

FPGA: Field Programmable Gate Arrays





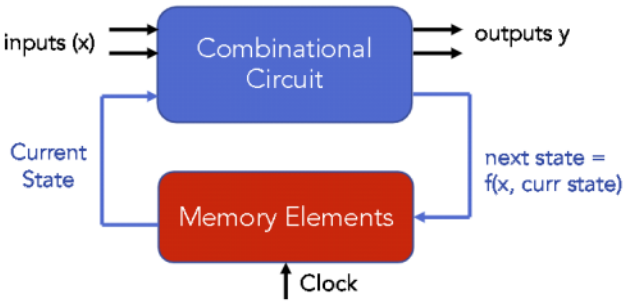


4-input LUT based BLE (Basic Logic Element)

SystemVerilog

Tuesday, April 5, 2022 5:01 PM

Combinational: Only boolean logic elements
Sequential: Stores a state which captures past history



Behavioral Model: Specifies the expected behavior of the system without regard for the gate level details

Dataflow Model: Describes how data flows through the system, uses logical expressions

Gatelevel Model: Models individual gates and interconnects, uses logic gate components, can specify delays

Transistor Level: Models each transistor, uses NMOS, PMOS, CMOS

Modules, Parameters, Ports

Thursday, April 7, 2022 5:05 PM

Def: A Module is a container that holds design behavior
- Can hold other modules

Format:

Module start and end declaration, contains module name
Optional parameter list
Primary port declarations
Local nets and variable declarations
Concurrent statements which define functionality
Instances of other modules

```
module counter              // Module name declaration
  #(parameter WIDTH=4)      // Parameter declaration
   (
      input logic clk,
      input logic clear,
      output logic [WIDTH-1:0] count
   );
 logic[WIDTH-1:0] cnt; // Local variable declaration

 always @(posedge clk or posedge clear)
   begin
     if (clear == 1)
       cnt = 0;
     else
       cnt = cnt + 1;
   end

 assign count = cnt;
endmodule: counter  // Module end declaration
```

Primary port declarations with directions and data type of each port specified

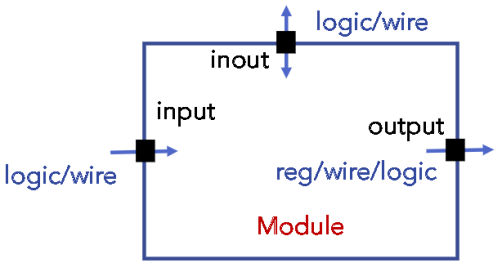
Functionality of design using concurrent statements

Parameters: Optional values with defaults
- Useful for making module configurable
- Can be overridden on instantiation

Ports: set of signals that act is inputs/outputs
Syntax: <direction> <datatype> <width> name;
Directions: input, output, inout
Datatype: logic, wire, reg
- wire: combinational driven value
- reg: sequential driven value
- Logic: either wire or reg

Width: example [N-1 : 0]

Defaults: input wire 1



- wire is a 4-state net
- reg is a 4-state variable
- logic is a 4-state net or a variable

Connecting Ports

Thursday, April 7, 2022 5:58 PM

```
// port order
wire a, b, c;
circuit inst1 (
    a,
    b,
    c
);
```

```
// explicit
wire a, b, c;
circuit inst1 (
    .b (b),
    .a (a),
    .c (c)
);
```

```
// dot-name
wire a, b, c;
circuit inst1 (
    .a,
    .b,
    .c
);
```

```
// dot-star
wire a, b, c;
circuit inst1 (
    .*
);
```

Port Order:

- Simple to code
- Error prone, difficult to debug, ORDER IS IMPORTANT

Explicit: Encouraged

- Verbose, each port needs to be specified
- Prevents accidents, order does not matter

Dot-Name:

- Less verbose, infers connections by matching names
- Potentially error prone, names must match

Dot-Star:

- Simplest
- Most error prone

Specifying Parameters

Thursday, April 7, 2022 6:05 PM

Parameters can be internal or external:

- Parameter is specified externally
- Localparam is specified internally

Syntax: parameter/localparam <datatype> <signedness> <size> name = <value>;

Parameters overridden using the defparam statement or:

```
half_adder #(.N(4)) ha_instance(
    .a(in1),
    .b(in2),
    .sum(sum),
);
```

Data Kinds, Data Types, Nets(Wire), Variables(Reg)

Tuesday, April 12, 2022 5:13 PM

Data is declared using the syntax: <data kind> <data type> <name>;

```
var logic carry_out;
wire logic [2:0] sum;
```

Data types are divided into two main groups: Nets and Variables

Net or Variable are 2 (0, 1) or 4 state variables (0, 1, X, Z)

Nets: represent a physical connection between structural entities

- Does not store value
- Is continuously driven
- Wire is most common Net

Wire elements are usually used by combinational logic

- Only legal on left hand side of assign statements

Variables: represent data storage element in circuit

- Stores some values for simulation, does not indicate actual storage in silicon
- Driven by sensitivity list (usually a clock)
- Reg is most common Variable

Reg elements models hardware registers

- Used in always@(…) blocks
- Only legal in left hand side of "=" or "<=" statements

Continuous Assignment

Tuesday, April 12, 2022 5:40 PM

Idea: Using the "assign" statement, drive multiple LHS Net type concurrently

Example:

```
assign sum = a^b;
assign cout = a & b;
```

Continuous assign cannot be used inside always or initial blocks

Continuous assignment can be inferred from always block
Example: always@(*)
Since the sensitivity list includes all inputs, then the statements can be interpreted as combinational

Conditional Operator

Tuesday, April 12, 2022 5:57 PM

An if else statement can be condensed into the Ternary Operator:

assign out = p ? a:b;

Conditional expression listed before **"?"** is evaluated first as true or false

- If evaluation result is true, then **true expression** is evaluated
- If evaluation result is false, then **false expression** is evaluated
- If evaluation result is unknown "x", then conditional operator performs bit by bit comparison of the two possible return values
  - If corresponding bits are both 0, a 0 is returned for that bit position
  - If corresponding bits are both 1, a 1 is returned for that bit position
  - If corresponding bits differ or if either has "x" or "z" value, an "x" is return for that bit position

Example:

```
logic sel, mode;
logic [3:0] a, b, mux_out;
assign mux_out = (sel & mode) ? a : b;
```

Scenario	Value of "sel"	Value of "mode"	Value of "a"	Value of "b"	Result of conditional expr (sel & mode)	Final value assigned to "mux_out"
1	1'b1	1'b1	4'b0101	4'b1110	True (1)	4'b0101
2	1'b0	1'b1	4'b0101	4'b1110	False (0)	4'b1110
3	1'b1	1'bx	4'b0101	4'b1110	Unknown (x)	4'bx1xx
4	1'b1	1'bx	4'b011x	4'b0z10	Unknown (x)	4'b0x1x

Blocking and Non-Blocking

Thursday, April 14, 2022 4:59 PM

Initial block models behavior which is applied only at t = 0

Always block describes behavior which continuously runs
- Has sensitivity list @(…) which determines when always block is triggered
  - Can model both combinational (always_comb or always @(*) or sequential (always @(posedge clock or negedge clock)
  - Also can detect level changes (posedge or negedge) using @(x)
- Uses blocking or non-blocking statements

Non-Blocking statement: f = a + b;
- Used to model combinational logic
- Executes multiple statements in order specified
- Values assigned immediately, blocks progression on each assignment

Blocking statement: f <= a + b;
- Used to model sequential logic
- Executes multiple statements concurrently
- Values assigned at the end of block, does not block progression on each assignment

Execution order:
1) Blocking statements, continuous assignments
2) #0 Blocking statements
3) Non-blocking statements updated

Guidelines:
- Always use blocking in always_comb
- Always use non-blocking in always_ff

Inter/Intra Delays

Tuesday, April 19, 2022 5:56 PM

Def: Inter assignment delay (delay on LHS)

- Execution of entire statement is delayed
- Syntax: #<delay> <LHS> = <RHS>

Def: Intra assignment delay (delay on RHS)

- Only RHS assignment is delayed
- Syntax: <LHS> = #<delay> <RHS>

Procedural Blocks

Thursday, April 21, 2022 5:07 PM

Def: Procedural Blocks are either initial blocks or always blocks.
Initial: runs once at t = 0
Always: runs once when sensitivity list is triggered

Initial blocks are non-synthesizable
Always blocks can be synthesized

always_comb, always_ff, always_latch

Thursday, April 21, 2022 5:12 PM

always_comb: used to model combinational logic
- Infers a complete sensitivity list from statements
- Executes once at t = 0
- Incomplete case statements not allowed
- Multiple drives not allowed

always_ff: used to model sequential flip-flop logic
- Sensitivity list must be specified as edge level (posedge or negedge) of clock and asynchronous set/reset signals
- Mixing of signed edge and double edge not allowed
- Sensitivity list cannot contain other signals like Data input or enable input
- Multiple drives not allowed
- Cannot mix blocking and not blocking statements

always_latch: used to model sequential latch flip-flip logic
- Infers a complete sensitivity list from statements
- Constructs such as #, @, wait which delay execution of statements are not allowed
- Multiple drivers not allowed

Initial blocks

Tuesday, April 26, 2022 6:11 PM

Idea: Used to develop testbench code to simulate RTL code

- Create how stimulus will be applied to signals
- Specifies initialization of local variables
- Specifies how the design signals are monitored (can dump to file)
- Specifies simulation pass and fail criteria
- Executed only once at the beginning of the simulation

Note: Multiple initial blocks can result in race conditions

If, Case

Thursday, April 28, 2022 5:01 PM

Decision Statements:

- If-else: inferred to be a mux
  - 1 is true, 0 x z are false
  - Use logical and not bitwise logical operators

- If (w/out else): inferred to be a latch
  - 1 is true, 0 x z are false
  - Use logical and not bitwise logical operators

- Case: inferred to be mux, decoder, encoder or next state logic (in FSM)
  - Has implied break statement at end of each item, used begin/end to create multiple line items
  - Incomplete case statements will infer a latch
  - Case items are not-necessarily non-overlapping

  - Case (…) inside: can use "?" as wildcard operators, bits with value 0, 1, X, Z will be ignored
  - Casez: wildcard "Z, ?"
  - Casex: wildcard "X, Z, ?"

  - "unique" modifier: indicates items can be evaluated in parallel, all items are unique
  - "priority" modifier: indicates items must be evaluated in order listed, items are not unique

Functions

Thursday, April 28, 2022 6:07 PM

Functions: describes block of combination logic operations

- Can be called multiple times, which enables reusability
- Can be called from continuous assignment, always, initial blocks
- Can be synthesizable or non-synthesizable depending on following coding guidelines
- Must have input, output, inout, logic ports declared in argument list
- Return: values can be returned by explicit "return" statement, or by assigning function name to value

```
function logic[1:0] add3(input logic [1:0] x, y, z);
  logic [1:0] t;
  begin
    t = x + y;
    return t + z;
  end
endfunction
```

Returning value of t+z using "return" keyword instead of assigning to implicit variable add3

```
function logic[1:0] add3(input logic [1:0] x, y, z);
  logic [1:0] t;
  begin
    t = x + y;
    add3 = t + z;
  end
endfunction
```

Returning value of t+z using by assigning result to implicitly declared variable name add3 which same as function name

  - Void: function has no direct return types, can drive ONE output argument

```
function void add3(input logic [1:0] x, y, z, output logic [1:0] sum);
  logic [1:0] t;
  begin
    t = x + y;
    sum = t + z;
  end
endfunction
```

Returning value of t+z by driving output variable sum

- Function types: Static, automatic - determine local variable scope
  - Static/automatic: static retains internal values every call, automatic clears local variables every call

```
function logic[1:0] add3(input logic [1:0] x, y, z);
  logic [1:0] t;
  begin
    t = x + y;
    add3 = t + z;
  end
endfunction
```

Variable "t" is shared across all invocations of "add3" since add3 is a static function !!!

```
function automatic logic[1:0] add3(input logic [1:0] x, y, z);
  logic [1:0] t;
  begin
    t = x + y;
    add3 = t + z;
  end
endfunction
```

"automatic" ensures that all local variables are truly local. Each invocation of "add3" will use a different "t"!

Generate

Tuesday, May 3, 2022 5:12 PM

Generate: used to instantiate decision based modules

- Can be used with if/else or for loop to instantiate modules based on decision logic or logic
- Permitted items in generate statements:
  - Modules, primitive instances
  - Initial or always procedural blocks
  - Continuous assignment
  - New and variable declarations
  - Parameter redefinitions
  - Task or function definitions
- Illegal items in generate statements:
  - Port declarations, constant declarations, specify blocks
- genvar: special iteration variable that is accessible only during elaboration, not accessible runtime
  - Can only be positive number, 0, X or Z

Looping Statements

Tuesday, May 3, 2022 5:28 PM

Looping Statements:

- For loop: used to expand hardware logic, iterations must be fixed number
- Ex: apply logic between consecutive bits of multiple bit input
- Repeat: execute block a set number of times, synthesizable but commonly used in testbenches
- While: executes block wile condition is true, synthesizable if bounds are fixed value
  - Condition is evaluated at top of loop body
- Do-while: while loop but will run at least once, synthesizable if bounds are fixed value
  - Condition is evaluated at end of loop body
- Forever: executes block indefinitely, not synthesizable
- Foreach: iterates through dimensions of unpacked array, not synthesizable
  - Will automatically declare loop control variables, exit conditions, increment/decrement

Tasks, Tasks vs Functions

Tuesday, May 3, 2022 5:31 PM

Tasks: encapsulates one or more statements which can be called from different parts of the code
- Syntactically similar to functions but do not have return values

Rules:
- Can be static (retains local variables for next call) or automatic (new calls have new local variables)
- Contain blocking and non-blocking assignments
- contain any time controlled statements such as #, @, or wait, posedge, negedge, etc
- call another tasks and functions
- have zero or multiple inputs and outputs specified in its argument list
- take, drive and source global variables, when no local variables are used
- be used in both synthesizable and non-synthesizable code

Tasks vs Functions:

Functions

- Cannot contain time-controlling statements (functions execute in "zero time")
- Can call other functions but cannot call a task
- Must have at least one input argument
- Returns a single value

Tasks

- Can contain time-controlling statements
- Can call other tasks and functions
- Arguments are optional
- Does not return values but can affect multiple values

Finite State Machines

Thursday, May 5, 2022 5:01 PM

Def: An FSM is combination of sequential state with combination logic feedback

- Must have finite number of states
- Can only be in one state at a time (current state/present state)
- Transitions between states by triggering event or condition
- Defined by initial state, list of states, and transition conditions

Mealy: output is determined only by present state

- Slower response time, output changes on next clock cycle
- 2 block approach:
  - Sequential logic to manage present state
  - Combinational logic to determine next state and outputs
- 3 block approach:
  - Sequential logic to manage present state
  - Combinational logic to determine next state
  - Combinational logic to determine outputs

Moore: output is determined by both present state and inputs

- Faster response time, output changes immediately with input change
- 2 Block approach:
  - Sequential logic to manage present state
  - Combinational logic to determine next state and outputs

1 block approach:

- Faster in simulation
- Cannot simulate mealy FSM

Note: Registering the output of mealy FSM is equivalent to a moore FSM

State Encoding

Tuesday, May 10, 2022 5:26 PM

Possible ways to encode states in a FSM:

- Binary value encoding: space efficient encoding but slower combinational logic
- One-hot encoding: space inefficient encoding but faster combinational logic
- Gray encoding
- Johnson encoding
- LFSM encoding

Declaring encodings in SystemVerilog

1. Using enumeration

```
enum logic[1:0] {WAIT=2'b00, EDGE=2'b01}
present_state, next_state;
```

2. Using parameter

```
parameter  logic [1:0]  WAIT = 2'b00, EDGE =
2'b01;
logic [1:0] present_state, next_state;
```

3. Using local parameter

```
localparam  logic [1:0]  WAIT = 2'b00, EDGE =
2'b01;
logic [1:0] present_state, next_state;
```

4. Using typedef (SystemVerilog only)

```
typedef enum logic [1:0] {WAIT=2'b00,
EDGE=2'b01} e_states;
e_states present_state, next_state
```

Cryptographic Hashing, SHA 256

Thursday, May 12, 2022 4:58 PM

Cryptographic Hashing should have the following properties:

- Compression: hash output should be the same length regardless of input size
- Avalanche Effect: small changes in the input should have large changes in the output
- Determinism: the same input should always have the same hash
- One-way function: Easy to hash but difficult to unhash
- Collision Resistance: Two inputs with the same hash should be extremely rare
- Efficient: Should be relatively quick to compute

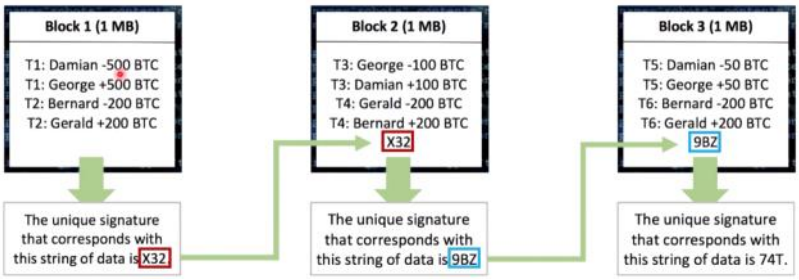
SHA-256:

- Message <= 2^64 bits
- Message processed in 512 bit blocks sequentially
- Hash value is 256 bits

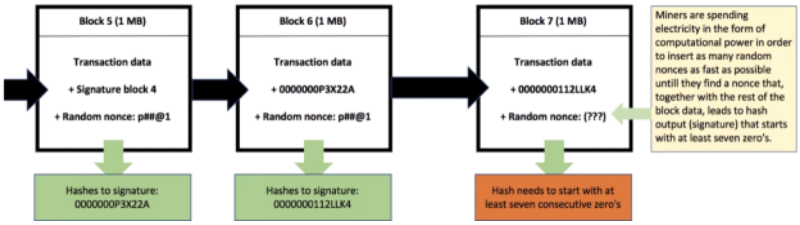
Bitcoin Hashing Algorithm

Monday, May 16, 2022 4:18 PM

Using SHA 256 hashing algorithm, we can keep a distributed ledger of transactions:



Transactions are inserted into the blockchain only if it starts with a specific pattern (ie 7 consecutive 0s):
The block will include a variable called the nonce which allows a block to follow the pattern



When a transaction is made, the sender calculates a nonce, and other miners compete to find the nonce to verify the transaction. Once the nonce is found, the transaction is added to the ledger

The system is secure because any malicious miner would have to out hash all other miners to catch up