ELEX 2117 : Digital Techniques 2
2021 Winter Term

HDL Idioms

After this lecture you should be able to convert a Verilog description into a block diagram, and convert a block diagram into a Verilog description.

Common HDL Idioms

Logic synthesizers such as Quartus convert HDL descriptions into circuits (actually, netlists). They do this by recognizing a small number of idioms.

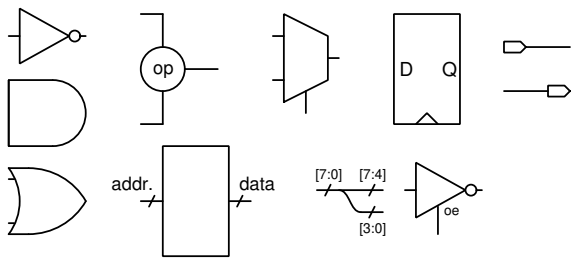
You must be able to visualize the hardware that would be generated by an HDL description in order create efficient designs. This lecture describes some common HDL constructs and their corresponding hardware implementations.

Combinational Logic

The following HDL constructs are typically synthesized as follows:

- logical operators are converted to logic gates as you would expect.
- arithmetic and comparison operators are converted to blocks of combinational logic implementing the required operation.
- access to a value in an unpacked array is a implemented as read-only memory (ROM). Access to bits in packed arrays is implemented as a connection to specific bits.
- ternary operators, if/else and case statements are converted into multiplexers; nested as necessary
- an output to which '`z`' can be assigned is converted to a tri-state output.

Exercise 1: Using the schematic symbols shown below, convert each of the following System Verilog expressions into a schematic.




```
// assume the following declarations:
logic a, b, c, reset, up, down, oe ;
logic [15:0] x, y, y_next ;
logic [3:0] z ;
logic [3:0] tab [64] ;


assign c = a ^ b ;


assign a = x < y ;


assign y_next = y + 1 ;


assign y = x[3] ;


assign y = 1'b1 << x ;  // decoder


assign z = tab[x] ;


assign y_next = y < x ? y + 1 : y - 1 ;


assign y = z[3] ? 4 :  // priority encoder
           z[2] ? 3 :
           z[1] ? 2 :
           z[0] ? 1 : 0;


assign y = oe ? x : 16'hzzzz ;


assign y_next = reset ? 16'h0 :
                up ? y + 1 :
                down ? y - 1 :
                y ;
```

Sequential Logic

The following HDL constructs are synthesized as follows:

- **`always_ff`** blocks with sensitivity lists containing signal edges (**`@(posedge...)`**) are converted to registers. The signals assigned to within the **`always_ff`** block are the register outputs.
- combinational logic that computes the values assigned to register inputs is used to describe

specialized sequential logic such as counters and shift registers. The register's current value is often used to compute the next value.

Exercise 2: Using the schematic symbols shown above, convert each of the following System Verilog expressions into a schematic.

```
// shift register
assign y_next = {y[14:0],a} ;
always_ff@(posedge clk)
   y = y_next ;


// FF showing if y (above) is even
assign a_next = y_next [15] ;
always_ff@(posedge clk)
   a = a_next ;


// register with load enable
assign y_next = en ? x : y ;
always_ff@(posedge clk)
   y = y_next ;


// counter with reset and count enable
assign y_next = reset ? '0 :
                en ? y + 1'b1 : y ;
always_ff@(posedge clk)
   y = y_next ;


// writeable memory (RAM)
always_ff@(posedge clk)
   tab[x] = z ;


// read port
assign z = tab[x] ;


assign y_next = reset ? 16'd7 :
                ( y == 16'd8000 && !down ) ? y-1 : y ;



// CRC
assign y_next = { ^( y & 16'h0x1021 ), y[15:1] } ;
always_ff@(posedge clk)
   y = y_next ;
```

Schematics to HDL

It's also important to be able to write the HDL that will result in a specific hardware architecture.

Each circuit element is converted into the corresponding HDL construct and named signals are used to connect them according to the circuit topology.

Exercise 3: Write System Verilog that would generate each of the following schematics. Include any required signal declarations (using `logic`).