ELEX 2117 : Digital Techniques 2
2020 Fall Term

HDL Idioms

This lecture describes more Verilog features for modelling hardware.
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.

More Verilog

Reserved Words

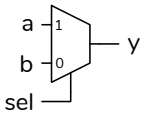
Use of reserved words as Verilog identifiers will cause syntax errors. These words include some that you may inadvertently use in your designs such as `reg`, `table` or `input`. An editor with syntax highlighting is a convenient way to identify and avoid reserved words.

Ternary Operator

Verilog's ternary operator is a concise syntax for describing a two-way multiplexer. The operator consists of three parts: the condition, the true value and false value. The result of the operator is the true value if the condition is non-zero, or the false value otherwise. For example:

```
assign y = sel ? a : b
```

⇔



would connect either `a` or `b` to `y` depending on the value of `sel`. Multiple ternary operators can be used to concisely describe trees of multiplexers.

Exercise 1: Draw the schematic corresponding to: `y = a ? ( b ? s1 : s2 ) : ( c ? s3 : s4 )`

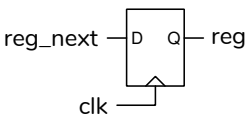
Single-Assignment `always_ff` Block

You should use the following structure for every register:

```
logic <width> reg, reg_next ;

always_ff@(posedge clk)
    reg = reg_next ;

assign reg_next = <expression> ;
```



This declares a signal with `_next` appended for each register (`reg` in this case). The `_next` signal is the input of that register. This may seem excessively verbose but: (a) it provides convenient access to both the current value (the output) and the next value (the input) of the register, (b) the use of two signals mimics the hardware and (c) one assignment per `always_ff` makes behaviour independent of their order in your HDL.

Exercise 2: Which signal is the next value of the register? When does it become the current state?

Arrays

Variables may have (multiple) "packed" and "unpacked" dimensions.

Packed dimensions are those given before the signal name. These bits are stored contiguously ("packed") and the packed item can be treated as a scalar – a single number – in expressions. Packed dimensions typically model a word or bit fields within a word. For example, `logic [3:0][7:0] ax ;` would describe a 32-bit word composed of 4 bytes of 8 bits and `ax[3]` would be the most-significant byte and `ax[0][7:4]` would be the most-significant nybble of the least-significant byte.

Unpacked dimensions appear after the signal name. These bits may not necessarily be stored contiguously. Unpacked dimensions model memories where only one element can be accessed at a time. For example: `logic [7:0] rom [32] ;` would model a 32-byte memory.

In array references, the unpacked dimension(s) are specified first, followed by the packed dimensions (if any). For example, `rom[31][0]` would be the least-significant bit of the last word in the `rom` above.

Slices and Concatenation

Part of a packed array (a "slice") can be referenced with a range of indices as shown above. Arrays can also be spliced back together with the concatenation

operator (`{,}`). For example, we can swap the bytes of a 16-bit word `b` using: `{b[7:0],b[15:8]}`.

Array literals (constants) can be defined by grouping the individual elements within `'{...}`. The quote distinguishes array literal syntax from the syntactically similar concatenation operator.

Logic Values

Four-state types, such as `logic`, can have four values: 0 (false), 1 (true), `x` (unknown) and `z` (high impedance). These are used for modeling logic.

A numeric constant can also include `x` (unknown) or `z` (high-impedance) values. These have useful interpretations for synthesis (don't-care and high-impedance respectively) but when used in simulations the result, in most cases, will be unknown (`x`).

The notations `'0` and `'1` are convenient abbreviations for a constant that is all-zeros or all-ones.

Tri-State Outputs

Tri-state outputs (those with high, low and high-impedance output states) can be modelled using `logic`. When such outputs are assigned the value '`z`' the output is placed in the high-impedance ("tri-stated") mode.

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.

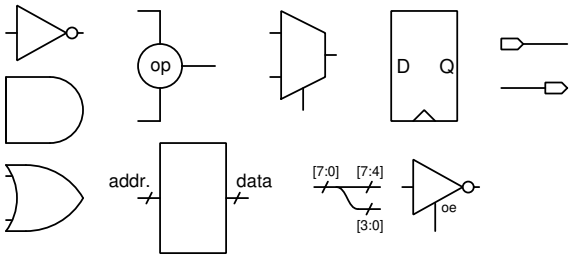
You must be able to visualize the hardware that would be generated by an HDL description in order create efficient designs. This section describes some common HDL constructs and their corresponding hardware implementations.

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




```
y = a ^ b ;
```

```
y = a < b ;
```

```
y = y+1 ;
```

```
y = a[3] ;
```

```
y = a[3] ? 4 : a[2] ? 3 : a[1] ? 2 :
    a[0] ? 1 : 0;
```

```
y = table[x] ;
```

```
if ( y < b )
  y = y+1 ;
else
  y = y-1 ;
```

```
y = oe ? d : 16'hzzzz ;
```

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

```
always_ff@(posedge clk)
  y = a ;
```

```
always_ff@(posedge clk)
  y[7:0] = {y[6:0],a} ;
```

```
always_ff@(posedge clk)
  if ( e )
     y = a ;
  else
     y = y ;
```

```
always_ff@(posedge clk)
  if ( r )
    y = '0 ;
  else
    if ( e )
       y = y+1'b1 ;
    else
       y = y ;
```

```
next = ( reset || done ) ? '0 : cnt+'b1 ;
```

```
always_ff@(posedge clk)
  if ( falling )
     mosi = sr[31] ;
```

```
always_ff@(posedge clk)
   cnt = cnt_next ;
```

```
// logic [31:0] mem [15:0]
always_ff@(posedge clk) begin
   mem[p] = din ;
```

```
// logic [31:0] mem [15:0]
dout = mem[p] ;
```

```
p_next = valid && rdy ? p + 1'b1 : p  ;
```

```
// i, j are logic[4:0]; w, sclk are logic
nxt = w ? 5'd7 : ( j==N && sclk ) ? i-1 : i ;
```

```
readdata = {31'b0,csn} ; // csn is logic
```

```
crc = ^ (g&sr) ; // g and sr are logic[31:0]
```

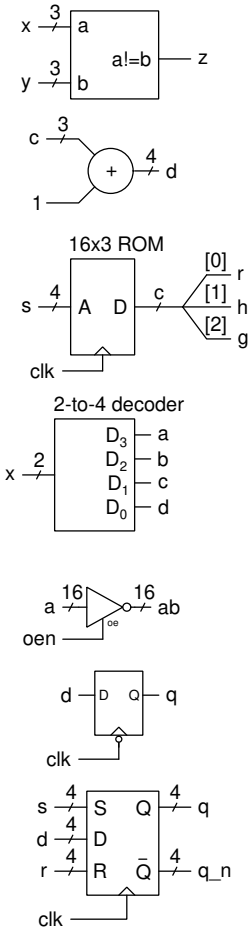
```
nxt = ~d[8] ;
```

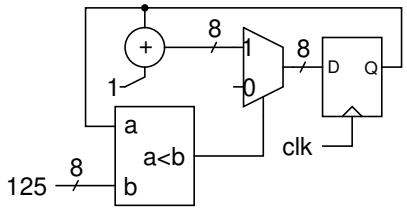
Schematics to HDL

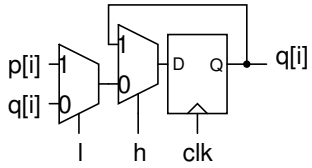
It's also important to be able to write the HDL that will result in a specific hardware architecture.

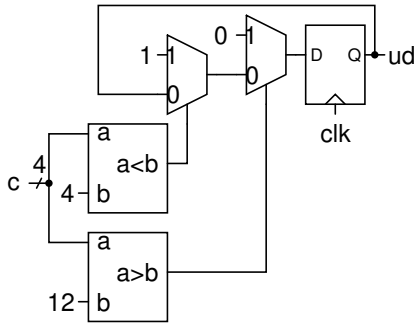
Each circuit element is converted into the corresponding HDL construct and named signals are used to connect them according to the circuit topology.

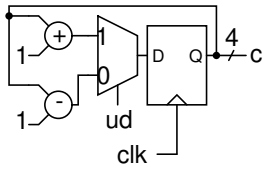
Exercise 5: Write System Verilog that would generate each of the following schematics. Include any required signal declarations (using `logic`).

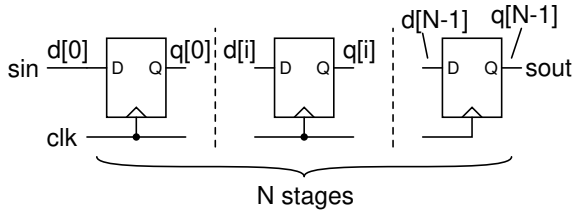



8
1
8
D Q
a
a<b
clk
125
8
b


p[i]
q[i]
q[i]
D Q
l
h
clk


1
0
0
1
D Q
ud
clk
a
a<b
c
4
4
b
a
a>b
12
b


1
1
0
D Q
4
c
1
ud
clk


d[N-1]
q[N-1]
sin
d[0]
q[0]
d[i]
q[i]
D Q
sout
clk
N stages