# HDL Idioms

*This lectures describes some common HDL constructs and their corresponding hardware implementations. 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.*

## 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.

## Common HDL Idioms

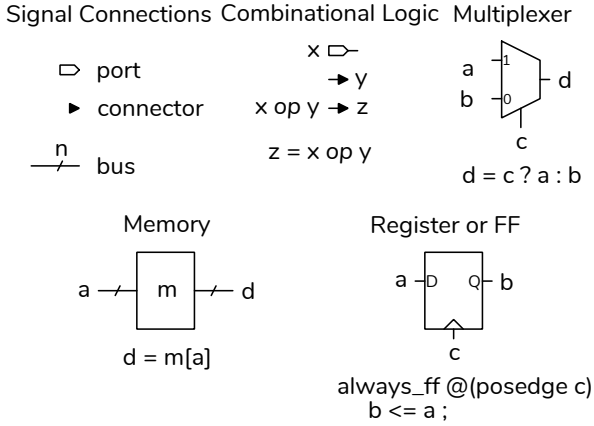
Logic synthesizers such as Quartus convert HDL descriptions into circuits (actually, netlists). They do this by recognizing a small number of idioms.

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.

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 [1] are converted into multiplexers; nested as necessary
- an output to which '`z`' can be assigned is converted to a tri-state output.
- `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 1:** Using the schematic symbols shown below, convert each of the following System Verilog expressions into a schematic.




[1] And if/else/case statements, when allowed.

```
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] ;
```

```
y = y < b ? y+1 : y-1 ;
```

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

**Exercise 2:** 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)
  y = y_next ;

assign y_next = e ? a : y ;
```

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

y_next = r ? '0 : e ? y+1'b1 : y ;
```

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

```
always_ff@(posedge clk)
    mosi = mosi_next ;

assign mosi_next = falling ? sr[31] : mosi ;
```

```
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
```

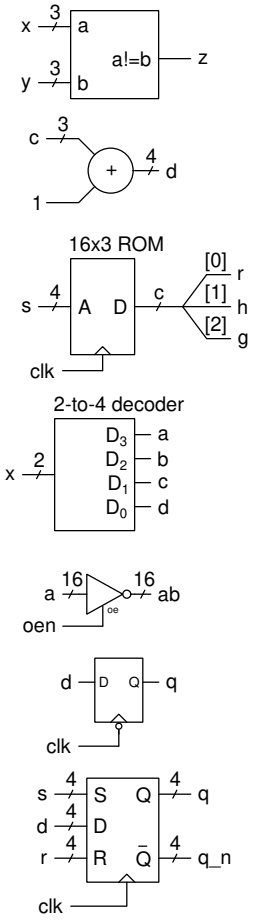
```
nxt = ~d[8] ;
```

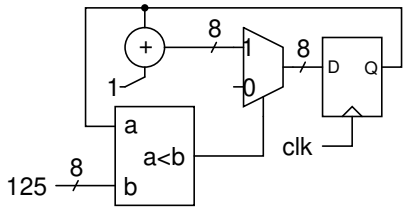
## Schematics to HDL

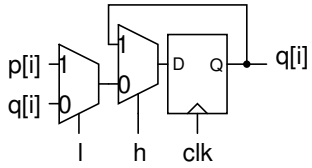
It's also important to be able to write the HDL that will result in a specific hardware architecture.

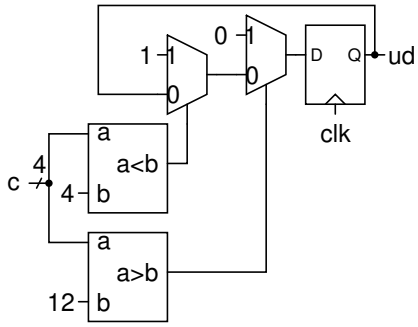
Each circuit element is converted into the corresponding HDL construct and named signals are used to connect them according to the circuit topology.

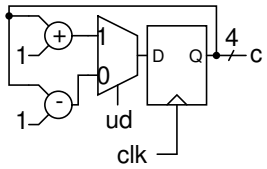
**Exercise 3:** Write System Verilog that would generate each of the following schematics. Include any required signal declarations (using **`logic`**).

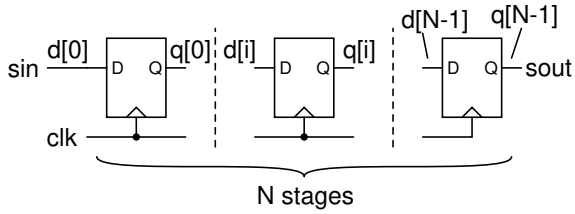



8
1
8
D Q
a
a<b
clk
125
8
b


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