Modules provide coarse-grain partitioning.

Now we need a technique for both fine-grain partitioning and to create the digital logic in those partitions.

This is done primarily with *always* blocks.

- *always_comb* creates strictly combinatorial logic
- *always_ff* creates flip flops only

In rare cases, we may want to describe latches.

- *always_latch* creates latches only
Within an `always_comb` block we can create

- AND/OR arrays of simple logic
- Arithmetic operations
- Comparison logic
- State machine next state steering logic

```verilog
always_comb begin
    unique case (done_sm_ps)
    NOT_DONE :
        begin
            done = 1'b0;  // not done indication
            if ((cycle_cnt==5'd31)&
                (mult_sm_ps==SHIFT)) done_sm_ns <= DONE;
            else             done_sm_ns <= NOT_DONE;
        end
    DONE :
        begin
            done = 1'b1;  // indicate done
            done_sm_ns <= NOT_DONE;  // go back
        end
    endcase
end
```
Within an `always_ff` block we create

- Just Flip-flops

```verilog
always_ff @(posedge clk, posedge reset)
  if (reset) done_sm_ps <= NOT_DONE; // at reset
  else done_sm_ps <= done_sm_ns; // else, goto next state
```

- FF's with the combinatorial logic that drives their D-inputs

```verilog
always_ff @(posedge clk, posedge reset)
  if (reset) cycle_cnt <= '0;
  else if(mult_sm_ps==SHIFT)
    cycle_cnt <= cycle_cnt + 1; // count up on shift
```

Any non-blocking assignment to a signal within an `always_ff` block creates a flip-flop whose Q output is connected to the name of the signal.
Note that:

- `always_comb` used the blocking """"="""" assignment
- *Use blocking assignments in always blocks that are written to generate combinational logic.*
- `always_ff` used the non-blocking """"<="""" assignment
- *Use nonblocking assignments in always blocks that are written to generate sequential logic.*

Ignoring these guidelines may infer correct logic gates, but pre-synthesis simulation might not match the behavior of the synthesized circuit.

---

System Verilog - Part 2

Let’s do some syntax cleanup...

- The ’ (apostrophe) is called the cast operator
- It’s used in several ways:
  - As in ”C” it can be used to force a size or type

```systemverilog
y = 10'(x-2);  // cast the value x-2 to be 10 bits in size
y = int'(2.0*3.0); // cast the value 6.0 to be the integer 6
y = signed'(x);  // cast x to be a signed variable
```

- It is also often used to denote a fill value. In other words it lets you set all the bits of a vector to a value without specifying radix or size.

```systemverilog
data = '1;  // set all the bits of data to V_dd}
if (rst) y1 <= '0;  // set all bits of y1 to V_ss (ground)
```