**Sequential Statements:**

**if-then-else**

*general format:*

```vhdl
if (condition) then do stuff
elsif (condition) then do more stuff
else do other stuff
end if;
```

*example:*

```vhdl
if (S = "00") then Z <= A;
elsif (S = "11") then Z <= B;
else Z <= C;
end if;
```

*elsif and else clauses are optional*

*BUT incompletely specified if-then-else (no else) implies memory element*

**case-when**

*general format:*

```vhdl
case expression is
  when value => do stuff
  when value => do more stuff
  when others => do other stuff
end case;
```

*example:*

```vhdl
case S is
  when "00" => Z <= A;
  when "11" => Z <= B;
  when others => Z <= C;
end case;
```

**for-loop**

*general format:*

```vhdl
for identifier in range loop
  do a bunch of junk
end loop;
```

*example:*

```vhdl
init: for k in N-1 downto 0 loop
  Q(k) <= '0';
end loop init;
```

*note: variable k implied in for-loop and does not need to be declared*

**while-loop**

*general format:*

```vhdl
while condition loop
  do silly stuff
end loop;
```

*example:*

```vhdl
init: while (k > 0) loop
  Q(k) <= '0';
  k := k – 1;
end loop init;
```

*note: variable k must be declared as variable in process (between sensitivity list and begin with format:*

```vhdl
variable k: integer := N-1;
```
Concurrent Statements:

**logical operators with signal assignment** $\leq$  
example: $Z \leq A \text{ and } B$;

**when-else**

general format:  
example:

$expression \text{ when } \text{condition} \text{ else }$  
$Z \leq A \text{ when } S = \text{“00”} \text{ else }$  
$B \text{ when } S = \text{“11”} \text{ else }$  
$C$;

*note: “when others” maybe redundant and incompatible with some tools*

**with-select-when**

general format:  
example:

$\text{with selection select }$  
$\text{with S select}$  
$Z \leq A \text{ when } \text{“00”},$  
$B \text{ when } \text{“11”},$  
$C \text{ when others};$

Signal assignments in a process:

All expressions based on current value of signals  
(right-hand side of $\leq$, values at start of process execution)  
Assigned signals updated at end of process execution

Example:

process (CK) begin  
$D \leq Q \text{ xor } CIN;$  
if (CK’event and CK = ‘1’) then  
$Q \leq D;$  
end if;  
$COUT \leq Q \text{ xor } CIN;$  
end process;

**Case 1:** sensitivity list consists only of CK (no other implied signals)  
on rising clock edge, Q gets value of D based on Q and CIN from previous execution of process  
if CIN is available prior to falling edge of CK then count works as expected otherwise, it does not  
also COUT is updated on falling edge of CK and not when Q changes

**Case 2:** sensitivity list consists of CK, CIN and Q (or CIN and Q implied)  
D and COUT updated anytime Q or CIN changes  
on rising clock edge, Q gets updated value of D & count works as expected
Signal assignments in a concurrent statement:
Like a process with implied sensitivity list (right-hand side of <=)
∴ multiple concurrent statements work like multiple 1-line processes
updates assigned signal whenever right-hand has event

Example:  
\[
\begin{align*}
D &= Q \text{ xor } CIN; \\
COUT &= Q \text{ xor } CIN; \\
\text{process (CK) begin} & \\
& \quad \text{if (CK’event and CK = ‘1’) then} \\
& \quad \quad Q = D; \\
& \quad \text{end if;} \\
& \text{end process;}
\end{align*}
\]

D and COUT updated anytime Q or CIN changes
on rising clock edge, Q gets updated value of D & count works as expected
same as if we put the 2 concurrent statements in process with Q & CIN in
sensitivity list

Initialization:
All processes (and concurrent statements) evaluated once
then again and again until there are no events in sensitivity list
If explicit initialization is not defined (using := assignment operator)
then a bit is assigned ‘0’ and a std_logic is assigned ‘U’
When no events happen for a given process sensitivity list then that
process is suspended

Simulation cycle:
1. Time is advanced until next entry in time queue where signals are to be
   updated (for example, PIs) which cause events on these signals
2. A simulation cycle starts at that point in time and processes & concurrent
   statements “sensitive” to events (during the current simulation time) on
   those signals will be executed
3. Simulation times for subsequent simulation cycles are determined and
   scheduled based on internal signals being updated from processes or
   concurrent statements
   Note: we will talk about specifying delays later, right now we consider only
delta (δ) delays = infinitesimal delays
4. If there are any δ delay time queues go to Step 2, else go to Step 1
   Examples:
   \[
   \begin{align*}
   Z &= X \text{ after 5ns;} \quad \text{-- specified delay scheduled as entry in time queue} \\
   Z &= X; \quad \text{-- δ delay scheduled as entry in δ delay time queue}
   \end{align*}
   \]