In order to design a sequential circuit, you need to use a logic element with memory – a flip-flop or a latch. AHDL has several types of such elements – a latch, a D flip-flop, a JK flip-flop, an SR flip-flop and a toggle flip-flop. Here we will discuss two of these elements — the latch and the D flip-flop.

**Latch** A latch in AHDL has two inputs – D and En, and one output Q. When En is low, the output Q does not change. When En is high, the output Q is equal to the input D. Thus, when En is low, it will hold the value on the D input when En went from high to low. To use a latch in AHDL, declare it in the VARIABLE section of the program:

```vhdl
VARIABLE
  A               : LATCH;
```

will define an eight-bit latch. To specify what should on the D input of A, use A.d. To specify what should on the En input of A, use A.ena. To use the Q output, refer to A.q.

**D flip-flop** There are two D-type flip-flops in AHDL – DFF and DFFE (D flip-flop with enable). DFF has the standard D flip-flop inputs and outputs – D input (D), clock input (CLK), active-low asynchronous clear input (CLRN), active-low asynchronous set input (prn), and the Q output (Q). DFFE has another input – enable (ENA). The ENA input to a DFFE must be high for the flip-flop to change state – with ENA low, the Q output will not change on a clock edge. To use a DFFE, declare it in the VARIABLE section of the program:

```vhdl
VARIABLE
  B               : DFFE;
```

Let’s build a simple 3-bit up counter: it will count 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, ... Here is the state transition table:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>y2 y1 y0</td>
<td>Y2 Y1 Y0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

This can be implemented with three D flip-flops with the following Boolean equations:
Y_2 = y_2y_0 + y_2y_1 + y_2y_1y_0
Y_1 = y_1y_0 + y_1y_0
Y_0 = y_0

Here is an AHDL program to implement a three-bit counter:

SUBDESIGN 3count
(  count[2..0] : OUTPUT;
    clock : INPUT;
)
BEGIN
VARIABLE
  y[2..0] : DFFE; % Three D flip-flops with enable %
END DEFAULTS;

  y[].ena = VCC; % flip-flops always enabled %
  y[].clrn = VCC; % flip-flops always enabled %
  y[].prn = VCC; % flip-flops always enabled %
END DEFAULTS;

  y[].clk = clock; % Use input clock to run flip-flops %
y2.d = (y2.q & !y0.q) # (y2.q & !y1.q) # (!y2.q & y1.q & y0.q);
y1.d = (!y1.q & y0.q) # (y1.q & !y0.q);
y0.d = !y0.q;
  count[] = y[].q; % Assign the outputs of the flip-flops to the output of the system %
END;

You can also design the counter by specifying the transition table and let AHDL determine the Boolean equations:

SUBDESIGN 3count
(  count[2..0] : OUTPUT;
    clock : INPUT;
)
BEGIN
VARIABLE
  y[2..0] : DFFE; % Three D flip-flops with enable %
END DEFAULTS;

  y[].ena = VCC; % flip-flops always enabled %
  y[].clrn = VCC; % flip-flops always enabled %
  y[].prn = VCC; % flip-flops always enabled %
END DEFAULTS;
y[].clk = clock; % Specify the clock for the D flip-flops 

TABLE

<table>
<thead>
<tr>
<th>B&quot;000&quot;</th>
<th>B&quot;001&quot;</th>
<th>B&quot;010&quot;</th>
<th>B&quot;011&quot;</th>
<th>B&quot;100&quot;</th>
<th>B&quot;101&quot;</th>
<th>B&quot;110&quot;</th>
<th>B&quot;111&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>=&gt;</td>
<td>=&gt;</td>
<td>=&gt;</td>
<td>=&gt;</td>
<td>=&gt;</td>
<td>=&gt;</td>
<td>=&gt;</td>
<td>=&gt;</td>
</tr>
<tr>
<td>B&quot;000&quot;</td>
<td>B&quot;001&quot;</td>
<td>B&quot;010&quot;</td>
<td>B&quot;011&quot;</td>
<td>B&quot;100&quot;</td>
<td>B&quot;101&quot;</td>
<td>B&quot;110&quot;</td>
<td>B&quot;111&quot;</td>
</tr>
<tr>
<td>B&quot;010&quot;</td>
<td>B&quot;011&quot;</td>
<td>B&quot;100&quot;</td>
<td>B&quot;101&quot;</td>
<td>B&quot;110&quot;</td>
<td>B&quot;111&quot;</td>
<td>B&quot;000&quot;</td>
<td></td>
</tr>
</tbody>
</table>

END TABLE;

count[] = y[].q; % Assign the outputs of the flip-flops to the output of the system 

END;

However, there is a much easier way to design counters. The inputs to the D flip-flops are the outputs of the D flip-flops plus one:

SUBDESIGN 3count
(   count[2..0] : OUTPUT;
    clock : INPUT;
)

VARIABLE

y[2..0] : DFFE; % Three D flip-flops with enable 

BEGIN

DEFAULTS

    y[].ena = VCC; % flip-flops always enabled 
    y[].clrn = VCC; % flip-flops always enabled 
    y[].prn = VCC; % flip-flops always enabled 

END DEFAULTS;

y[].clk = clock; % Specify the clock for the D flip-flops 

y[].q = y[].d + 1; % Next count is current count plus one 

count[] = y[].q; % Assign the outputs of the flip-flops to the output of the system 

END;
This gives you the ability to design very large counters which would be hard to do using other techniques. A 16-bit counter has $2^{16}$ or 65,536 states. It is difficult to develop the Boolean equations, and impractical to enter a transition table with 65,536 lines. Here is a design for a 16-bit counter:

```
SUBDESIGN 16count
(
  count[15..0] : OUTPUT;
  clock : INPUT;
)

VARIABLE
  y[15..0] : DFFE; % Sixteen D flip-flops with enable %

BEGIN
  DEFAULTS
    y[].ena = VCC; % flip-flops always enabled %
    y[].clrn = VCC; % flip-flops always enabled %
    y[].prn = VCC; % flip-flops always enabled %
  END DEFAULTS;

  y[].clk = clock; % Specify the clock for the D flip-flops %
  y[].d = y[].q + 1; % Next count is current count plus one %
  count[] = y[].q; % Assign the outputs of the flip-flops to the output of the system %

END;
```

Another, more powerful, way to design sequential circuits is with state machines. We will discuss how to do this in Lab 4.