A Toggle Flip-Flop Counter

As our final example, Listing 6.14 shows a 3-bit counter built from toggle flip-flops. We begin with line 7 which declares 3 DFFs T0, T1, and T2. These D flip-flops are transformed into toggle flip-flops by loading them with the inverse of their contents (line 11). Then the least-significant bit of the counter is clocked by positive-going edges of the input clock, C, on line 12. On line 13, the next bit of the counter is clocked by T0, the output of the LSB. In the same manner, the most-significant bit of the counter, T2, is clocked by T1 on line 14. Finally, on line 15 we provide an active-high asynchronous clear to each TFF so we can clear the counter at any time if needed.

Listing 6.14 ABEL code for a 3-bit counter built from toggle flip-flops.

```ABEL
001 MODULE TGLCNT
002 TITLE '3-bit toggle flip-flop counter'
003
004 DECLARATIONS
005 C PIN; "clock input"
006 ACLR PIN; "clear input"
007 T2..TO PIN ISTYPE 'REG'; "DFFs for counter"
008
009
010 EQUATIONS
011 [T2..TO] := ![T2..TO]; "make DFFs into TFFs"
012 T0.CLK = C; "LS bit clocked by input clock"
013 T1.CLK = T0; "remaining bits clocked by output"
014 T2.CLK = T1; "of the preceding bit"
015 [T2..TO].ACLR = ACLR; "connect the asynchronous clear"
016
017 END TGLCNT
```

The equivalent VHDL code for the 3-bit counter is shown in Listing 6.15. The toggle flip-flops are implemented as 3 different processes. On the rising edge of the main clock input, the bit0 process (lines 27–35) toggles the t0 TFF output by loading it with its inverse (lines 32 and 33). The t0 output is the clock for the next bit of the counter described by the bit1 process (lines 37–45). And the t1 output from that bit is the clock for the most significant bit of the counter described by the bit2 process (lines 47–55). Each process will override the toggling and clear its bit if the asynchronous clear input is high.

The VHDL code in Listing 6.15 also has special modifications so it will run on the XS400 Board. When targeting an XC4000 FPGA, the Foundation Series synthesis software tries to use special clock buffers to drive clock signals to the flip-flops. These buffers minimize the skew between the times at which the
clock edges reach the individual flip-flops. The Foundation Series Implementation tools also like to have the clock input come into the XC4000 FPGA through special pins which connect directly to these clock buffers. But for our example designs we use XSPORT to drive clock signals through the parallel port interface, and none of these pins directly connects to these special clock input pins. So we have to explicitly describe the connection of the clock input to one of the clock buffers. This is done on lines 24 and 25. On line 24, the main clock input is connected to an input buffer (IBUF) module. Then the output of the IBUF module is connected to the input of a general-purpose clock buffer (BUFG). The output of the BUFG, $c_{buf}$, becomes the clock input to the least-significant counter bit. (The IBUF and BUFG interfaces are described by the COMPONENT statements on lines 18–21.) You must use this clock input buffer circuitry when you are targeting the XC4000 FPGAs. For the XS95 Board and the XC9500 CPLDs, you can remove these buffers and use the clock input directly.

The simulated waveforms for this counter are shown in Figure 6.13. A high level on the ACLR input resets all the counter bits to zero. A series of rising edges on C moves the counter bits through the sequence 111 (7), 110 (6), 101 (5), 100 (4), 011 (3), 010 (2), 001 (1), 000 (0), 111 (7), and 110 (6). Then a logic 1 on the asynchronous clear input at 1.1 $\mu$s brings the counter back to 000 (0). All the features of the counter appear to be working.

A user-constraint file for the TGLCNT design should assign the following inputs and outputs for the XS40 Board:

```
NET c LOC=P44; # BO argument of XSPORT
NET aclr LOC=P45; # B1 argument of XSPORT
NET t0 LOC=P25; # S0 LED segment
NET t1 LOC=P26; # S1 LED segment
NET t2 LOC=P24; # S2 LED segment
```

Or use the following constraints for the XS95 Board:

```
NET c LOC=P46; # BO argument of XSPORT
NET aclr LOC=P47; # B1 argument of XSPORT
NET t0 LOC=P21; # S0 LED segment
NET t1 LOC=P23; # S1 LED segment
NET t2 LOC=P19; # S2 LED segment
```

For this experiment, the clock input, C, must be assigned to either B0 or B1. These inputs pass through a Schmitt-trigger buffer on the XS40 and XS95 Boards that removes any spurious glitches caused by the slow transitions of the PC parallel port signals. Using any of the B2–B7 inputs may cause the counter to decrement erratically.
Listing 6.15  VHDL code for a 3-bit counter built from toggle flip-flops.

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY tglcnt IS
  PORT  (
c: IN STD_LOGIC; -- clock
 aclr: IN STD_LOGIC; -- asynchronous active-high clear
t0: BUFFER STD_LOGIC; -- counter bit 0
t1: BUFFER STD_LOGIC; -- counter bit 1
t2: BUFFER STD_LOGIC -- counter bit 2
  );
END tglcnt;

ARCHITECTURE tglcnt_arch OF tglcnt IS
  COMPONENT IBUF PORT(I:IN STD_LOGIC;
                      O:OUT STD_LOGIC); END COMPONENT;
  COMPONENT BUFG PORT(I:IN STD_LOGIC;
                      O:OUT STD_LOGIC); END COMPONENT;
  SIGNAL c_in, c_buf: STD_LOGIC;-- buffered clock signals
  BEGIN
    uO: IBUF PORT MAP(I=>c,O=>c_in); -- bring clock through IBUF
    u1: BUFG PORT MAP(I=>c_in,O=>c_buf); -- route it through buffer

    bit0: PROCESS (c_buf,t0,aclr) -- process for counter LSB
      BEGIN
      IF aclr='1' THEN -- async clear bit if clear is high
        t0 <= '0';
      ELSIF (c_buf'event AND c_buf='1') THEN
        t0 <= NOT t0;
      END IF;
    END PROCESS bit0;

    bit1: PROCESS (t0,t1,aclr) -- process for middle counter bit
      BEGIN
      IF aclr='1' THEN
        t1 <= '0';
      END IF;
    END PROCESS bit1;
```

Experimental 199
Listing 6.15  VHDL code for a 3-bit counter built from toggle flip-flops.

```
041- -- toggle this bit when the LSB toggles from 0 to 1
042- ELSIF (t0'event AND t0='1') THEN
043-   t1 <= NOT t1;
044- END IF;
045- END PROCESS bit1;
046-
047- bit2: PROCESS (t1,t2,aclr) -- process for counter MSB
048- BEGIN
049- IF aclr='1' THEN
050-   t2 <= '0';
051- -- toggle the MSB when the middle bit toggles from 0 to 1
052- ELSIF (t1'event AND t1='1') THEN
053-   t2 <= NOT t2;
054- END IF;
055- END PROCESS bit2;
056-
057- END tglcnt_arch;
058-
```

Figure 6.13  Simulated waveforms 3-bit counter built from toggle flip-flops.
The following sequence of commands will step the downloaded design through the same set of states as the simulation:

<table>
<thead>
<tr>
<th>Command</th>
<th>ACLR</th>
<th>C</th>
<th>LED S2 (T2)</th>
<th>LED S1 (T1)</th>
<th>LED S0 (T0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSPORT 10</td>
<td>1</td>
<td>0</td>
<td>Dark (0)</td>
<td>Dark (0)</td>
<td>Dark (0)</td>
</tr>
<tr>
<td>XSPORT 00</td>
<td>0</td>
<td>0</td>
<td>Dark (0)</td>
<td>Dark (0)</td>
<td>Dark (0)</td>
</tr>
<tr>
<td>XSPORT 01</td>
<td>0</td>
<td>1</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
</tr>
<tr>
<td>XSPORT 00</td>
<td>0</td>
<td>0</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
</tr>
<tr>
<td>XSPORT 01</td>
<td>0</td>
<td>0</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
<td>Dark (0)</td>
</tr>
<tr>
<td>XSPORT 00</td>
<td>0</td>
<td>0</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
<td>Dark (0)</td>
</tr>
<tr>
<td>XSPORT 01</td>
<td>0</td>
<td>1</td>
<td>Bright (1)</td>
<td>Dark (0)</td>
<td>Bright (1)</td>
</tr>
<tr>
<td>XSPORT 00</td>
<td>0</td>
<td>0</td>
<td>Bright (1)</td>
<td>Dark (0)</td>
<td>Dark (0)</td>
</tr>
<tr>
<td>XSPORT 01</td>
<td>0</td>
<td>1</td>
<td>Bright (1)</td>
<td>Dark (0)</td>
<td>Dark (0)</td>
</tr>
<tr>
<td>XSPORT 00</td>
<td>0</td>
<td>0</td>
<td>Dark (0)</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
</tr>
<tr>
<td>XSPORT 01</td>
<td>0</td>
<td>1</td>
<td>Dark (0)</td>
<td>Bright (1)</td>
<td>Dark (1)</td>
</tr>
<tr>
<td>XSPORT 00</td>
<td>0</td>
<td>0</td>
<td>Dark (0)</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
</tr>
<tr>
<td>XSPORT 01</td>
<td>0</td>
<td>1</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
<td>Bright (1)</td>
</tr>
</tbody>
</table>

**Projects**

1. Change the edge-triggered DFF so it has an asynchronous preset input that sets it to a logic 1.

2. Do a unit-delay simulation of the edge-triggered DFF. Do you see where the setup and hold times arise?