ENGR 2323 Digital Design Lab
Counters and Clock Division

Counters

Counters are sequential circuits that generate an ordered sequence. An n-bit counter can count in binary from 0 to 2^n–1. For example, a 3-bit counter counts from 0 to 7 and the repeats. Assuming the state memory initially holds 000, the count sequence is 000, 001, 010, 011, 100, 101, 110, 111, 000 etc. Either n flip-flops or a n-bit register is needed for the state memory of a n-bit counter.

A modulo p counter counts from 0 to p-1 and then repeats the sequence. An n-bit counter is a modulo p counter where $p = 2^n$. For values of p that are not an integer power of 2, one could start with a n-bit counter design and add circuitry to reset the count to 0 when the count of p-1 was detected. A BCD counter (modulo 10) can be created from a 4-bit counter. The binary count sequence for a modulo 10 counter is: 0000, 0001, 0010 …, 1000, 1001, 0000.

The VHDL of Figures 1 and 3 show a modulo 3 counter behavioral design. The modulo 3 counter has three states for the counts from 0 to 2 and the state transitions go through the sequence of states corresponding to the count and then roll over and repeats when the count hits 2.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY counter IS
PORT(CLOCK, RESETN, CE : IN STD_LOGIC;
       F : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END counter;
```

Figure 1. Entity of Modulo 3 Counter

The functional simulation of the design is shown in Figure 2. The simulation shows counting through the full sequence and repeating after count 2 along with the operation when reset and when the count is not enabled.

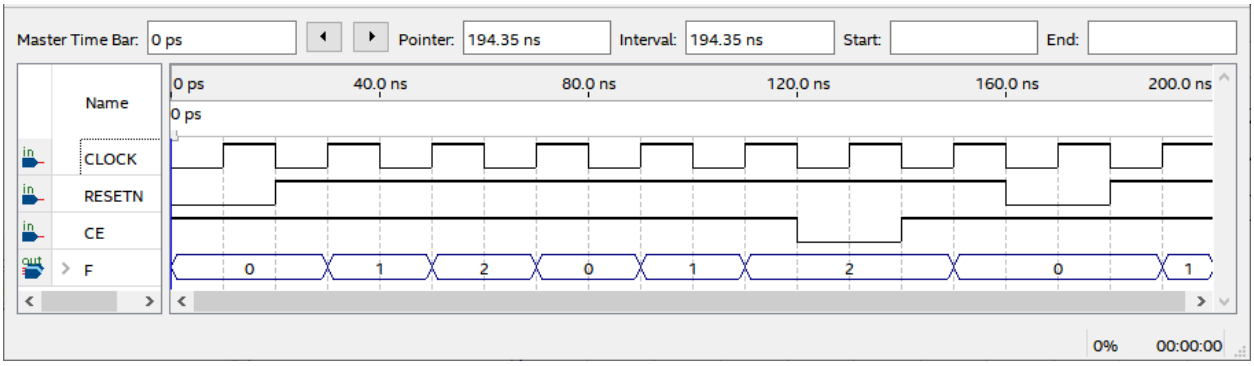

Figure 2. Functional Simulation of Modulo 3 Counter

```
ARCHITECTURE behavior of counter IS
TYPE STATETYPE IS (COUNT0, COUNT1, COUNT2);
SIGNAL STATE : STATETYPE;
BEGIN
   -- next state logic
   PROCESS(CLOCK, RESETN)
      BEGIN
          IF RESETN = '0' THEN
               STATE <= COUNT0;
          ELSIF RISING_EDGE(CLOCK) THEN
               CASE STATE IS
                    WHEN COUNT0 =>
                         IF CE = '1' THEN
                              STATE <= COUNT1;
                         ELSE
                              STATE <= COUNT0;
                         END IF;
                    WHEN COUNT1 =>
                         IF CE = '1' THEN
                              STATE <= COUNT2;
                         ELSE
                              STATE <= COUNT1;
                         END IF;
                    WHEN COUNT2 =>
                         IF CE = '1' THEN
                              STATE <= COUNT0;
                         ELSE
                              STATE <= COUNT2;
                         END IF;
               END CASE;
        END IF;
   END PROCESS;

   -- output logic
    F  <= "00" WHEN STATE = COUNT0 ELSE
      "01" WHEN STATE = COUNT1 ELSE
      "10" WHEN STATE = COUNT2 ELSE
      "00";

END behavior;
```

Figure 3. Architecture of Modulo 3 Counter

An alternative behavioral design for the modulo 3 counter is shown in Figure 4. The entity is the same as that used for the previous design, but instead of specifying the sequence of states the counter goes through, unsigned binary counting is described.

A 2-bit unsigned signal COUNT is defined. On reset, COUNT is initialized to zero (2-bit 00) and when counting, COUNT is incremented by one. When a count of 2 is detected, the count is reset to 0. To support unsigned arithmetic, the unsigned data type, and type conversion from unsigned to standard logic, the numeric package (`ieee.numeric_std`) is used. This package is also part of the IEEE library and provides support for unsigned and signed integers, arithmetic operations, comparisons, type conversion, etc.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY counter IS
PORT(CLOCK, RESETN, CE : IN STD_LOGIC;
       F : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END counter;

ARCHITECTURE behavior of counter IS
-- 2-bit unsigned number
SIGNAL COUNT : UNSIGNED(1 DOWNTO 0);

BEGIN
   PROCESS(CLOCK, RESETN)
      BEGIN
          IF RESETN = '0' THEN
               COUNT <= "00";
          ELSIF RISING_EDGE(CLOCK) THEN
               IF CE = '1' THEN
                    IF COUNT = "10" THEN
                         COUNT <= "00";
                    ELSE
                         COUNT <= COUNT + 1;
                    END IF;
               END IF;
          END IF;

     END PROCESS;

   -- convert unsigned count to standard logic
    F  <= STD_LOGIC_VECTOR(COUNT);

END behavior;
```

Figure 4. Modulo 3 Counter

For synthesis, the output F needs to be standard logic and the COUNT is thus converted from unsigned to standard logic. The functional simulation of the is design is identical to that shown in Figure 2.

The style of counter design shown in Figure 4 is useful for large counts where the number of states is prohibitively large. The design algorithm is at a higher level of abstraction than that of Figures 1 and 3 and realized more on the VHDL compiler for synthesis.

Clock Division

The DE10-Standard board has several external clocks (external to the Cyclone V FPGA) that can be used for designs. There are four 50MHz clocks for user logic, one 25MHz clock for the Hard Processor System (HPS), one 25MHz clock for ethernet, and two 24MHz clocks for USB.

For many practical applications, a lower clock frequency is needed. Phase lock loops (PLL) provide one mechanism for increasing or decreasing a clock frequency. The Cyclone V chip on the DE10-Standard board has six PLLs. A PLL may not be able to provide the specific frequency for an application so a clock division circuit may be used in addition to or instead of a PLL. For example, 1Hz is a common frequency in the design of a timer or other design where an actual time clock is required.

Creating a lower clock frequency from a higher clock frequency is clock division. The new frequency f_{new} is related to the original frequency f by a scale factor (divider) N.
$f/f_{new} = N$ or $f_{new} = f/N$

To create a 1kHz clock from a 50MHz clock, the scale factor N would need to be
N = 50M/1k = 50,000,000/1000 = 50,000. Note PLLs can also be used to provide a faster clock and support both multiplier and divider scale factors.

Figure 5 shows an example of clock division where the original clock is 25MHz (40ns period) and the output clock frequency after clock division is 6.25MHz (160ns period). A scale factor of 4 is used for the clock division. The VHDL code of Figure 6 was used to perform this clock division.

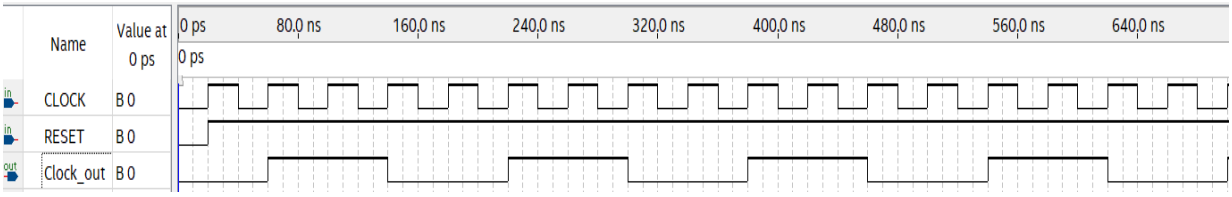

Figure 5. Clock Division

Like the example of Figure 4, the numeric package (`ieee.numeric_std`) is used to provide support for arithmetic and comparison operations. An integer signal COUNT is used to count the number of clock cycles of the input clock. Although the scale factor is 4, to achieve a 50% duty cycle of the output clock (equal time high and low in every period) , a count of 2 is detected and used to toggle the TEMP_CLOCK signal. Since TEMP_CLOCK is toggled twice every

output clock cycle, the scale factor is 4 (four input clock cycles for every one output clock cycle). Concurrent with the counting and toggling process, the CLOCK_OUT signal is set to the TEMP_CLOCK. The TEMP_CLOCK signal is needed since it is used on both the left and right side of signal assignment in the process (OUT signals can only be used on left side of signal assignment).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY clock_div IS
PORT(CLOCK_IN : IN STD_LOGIC;
   RESETN : IN STD_LOGIC;
   CLOCK_OUT : OUT STD_LOGIC);
END clock_div;

ARCHITECTURE behavior of clock_div IS
SIGNAL COUNT : INTEGER := 1;
SIGNAL TEMP_CLOCK : STD_LOGIC := '0';

BEGIN
   PROCESS (CLOCK_IN, RESETN)
   BEGIN
      IF (RESETN = '0') THEN
           COUNT <= 1;
           -- TEMP_CLOCK initialized with 0
           TEMP_CLOCK <= '0';
      ELSIF (RISING_EDGE(CLOCK)) THEN
           COUNT <= COUNT + 1;
           IF (COUNT = 2) THEN
                -- TEMP_CLOCK inverted every two cycles
                TEMP_CLOCK <= NOT(TEMP_CLOCK);
                COUNT <= 1;
           END IF;
      END IF;
   END PROCESS;

CLOCK_OUT <= TEMP_CLOCK;

END behavior;
```

Figure 6. VHDL for Clock Division with Scale Factor of 4

Using Clock Division as Part of a Design

A slower clock or timing operation may be needed for a design. These can be incorporated using single file or multiple file design. For a single file design, a VHDL architecture may have multiple process statements executing concurrently (in parallel). For example, a design

requiring a D flip-flop with a slower clock than available could have a clock division design and flip-flop design in parallel as shown in Figure 7.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dff_with_clock_div IS
PORT(D, CLOCK : IN STD_LOGIC;
   RESETN : IN STD_LOGIC;
   Q : OUT STD_LOGIC);
END dff_with_clock_div ;

ARCHITECTURE behavior of dff_with_clock_div IS
SIGNAL COUNT : INTEGER := 1;
SIGNAL TEMP_CLOCK : STD_LOGIC := '0';
BEGIN
   -- clock division
   PROCESS (CLOCK, RESETN)
   BEGIN
      IF (RESETN = '0') THEN
           COUNT <= 1;
           TEMP_CLOCK <= '0';
      ELSIF (RISING_EDGE(CLOCK)) THEN
           COUNT <= COUNT + 1;
           IF (COUNT = 2) THEN
                TEMP_CLOCK <= NOT(TEMP_CLOCK);
                COUNT <= 1;
           END IF;
      END IF;
   END PROCESS;

   -- D flip-flop using slower clock
   PROCESS (TEMP_CLOCK, RESETN)
   BEGIN
      IF (RESETN = '0') THEN
           Q <= '0';
      ELSIF (RISING_EDGE(TEMP_CLOCK)) THEN
           Q <= D;
      END IF;
   END PROCESS;

END behavior;
```

Figure 7. Combined D flip-flop Clock Divider Design

The architecture has two process statements, one describing the clock division operation and the other describing the D flip-flop operation. The D flip-flop uses the slower clock TEMP_CLOCK produced by clock division rather than the CLOCK input.

Single file designs often have simpler project structures but require the entity to support all the input and outputs signals for all portions of the design. Design reuse is more difficult with single file designs. A multiple file design often uses a schematic or structural VHDL design for the top level where all the designs are integrated. Multiple file designs are better for design reuse but the project structure is more complex.

References

[1] DE10-Standard User Manual, Terasic Inc., 2017

Last modified Tuesday, October 4, 2022