Chapter 10: Modeling Counters

Counters are a special case of finite state machines because they move linearly through their discrete states (either forward or backward) and typically are implemented with state-encoded outputs. Due to this simplified structure and widespread use in digital systems, VHDL allows counters to be modeled using a single process and with arithmetic operators (i.e., + and −). This enables a more compact model and allows much wider counters to be implemented. This chapter will cover some of the most common techniques for modeling counters.

Learning Outcomes—After completing this chapter, you will be able to:

10.1 Design a behavioral model for a counter using a single process.
10.2 Design a behavioral model for a counter with enable and load capability.

10.1 Modeling Counters with a Single Process

10.1.1 Counters in VHDL Using the Type UNSIGNED

Let's look at how we can model a 4-bit, binary up counter with an output called *CNT*. First, we want to model this counter using the "+" operator. Recall that the "+" operator is not defined in the std_logic_1164 package. We need to include the numeric_std package in order to add this capability. Within the numeric_std package, the "+" operator is only defined for types signed and unsigned (and not for std_logic_vector), so the output CNT will be declared as type unsigned. Next, we want to implement the counter using a signal assignment in the form *CNT <= CNT + 1*; however, since CNT is an output port, it cannot be used as an argument (right hand side) in an operation. We will need to create an internal signal to implement the counter functionality (i.e., CNT_tmp). Since a signal does not contain directionality, it can be used as both the target and an argument of an operation. Outside of the counter process, a concurrent signal assignment is used to continuously assign CNT_tmp to CNT in order to drive the output of the system. This means that we need to create the internal signal CNT_tmp of type unsigned also to support this assignment. Example 10.1 shows the VHDL model and simulation waveform for this counter. When the counter reaches its maximum value of "1111," it rolls over to "0000" and continues counting because it is defined to only contain 4 bits.

B. J. LaMeres, *Quick Start Guide to VHDL*, https://doi.org/10.1007/978-3-030-04516-6_10

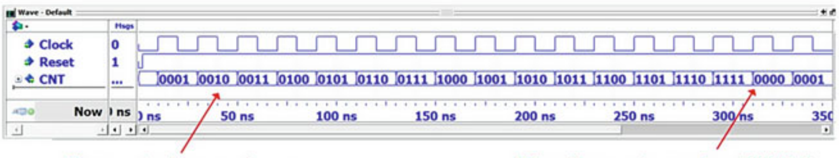
Example: 4-Bit Binary Up Counter in VHDL Using the Type UNSIGNED

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Counter_4bit_Up is
 port (Clock, Reset : in  std_logic;
       CNT          : out unsigned(3 downto 0));
end entity;

architecture Counter_4bit_Up_arch of Counter_4bit_Up is

 signal CNT_tmp : unsigned(3 downto 0);

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_tmp <= "0000";
      elsif (Clock'event and Clock='1') then
        CNT_tmp <= CNT_tmp + 1;
      end if;
  end process;

  CNT <= CNT_tmp;

end architecture;
```

The numeric_std package is needed to include the "+" operator. This operator only works on types signed/unsigned, so we will define the output CNT as type unsigned.

An internal signal is needed to support assignments in the form C <= C+1; because a port cannot be used as an argument in a signal assignment.

A concurrent signal assignment is used to continually assign CNT_tmp to CNT.



The counter increments on each rising edge of clock.

When the counter reaches "1111", it rolls over to "0000" and continues.

Example 10.1
4-Bit binary up counter in VHDL using the type UNSIGNED

10.1.2 Counters in VHDL Using the Type INTEGER

Another common technique to model counters with a single process is to use the type integer. The numeric_std package supports the "+" operator for type integer. It also contains a conversion between the types integer and unsigned/signed. This means a process can be created to model the counter functionality with integers and then the result can be converted and assigned to the output of the system of type unsigned. One thing that must be considered when using integers is that they are defined as 32-bit, two's complement numbers. This means that if a counter is defined to use integers and the maximum range of the counter is not explicitly controlled, the counter will increment through the entire range of 32-bit values it can take on. There are a variety of ways to explicitly bound the size of an integer counter. The first is to use an if/then clause within the process to check for the upper limit desired in the counter. For example, if we wish to create a 4-bit binary counter, we will check if the integer counter has reached 15 each time through the process. If it has, we will reset it to zero. Synthesizers will recognize that the integer counter is never allowed to exceed 15 (or "1111" for an unsigned counter) and remove the unused bits of the integer type during implementation (i.e., the remaining 28-bits). Example 10.2 shows the VHDL model and simulation waveform for this implementation of the 4-bit counter using integers.

Example: 4-Bit Binary Up Counter in VHDL Using the Type INTEGER

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Counter_4bit_Up is
 port (Clock, Reset : in  std_logic;
       CNT          : out unsigned(3 downto 0));
end entity;

architecture Counter_4bit_Up_arch of Counter_4bit_Up is

 signal CNT_int : integer;

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_int <= 0;
      elsif (Clock'event and Clock='1') then

        if (CNT_int = 15) then
          CNT_int <= 0;
        else
          CNT_int <= CNT_int + 1;
        end if;

      end if;
  end process;

  CNT <= to_unsigned(CNT_int, 4);

end architecture;
```

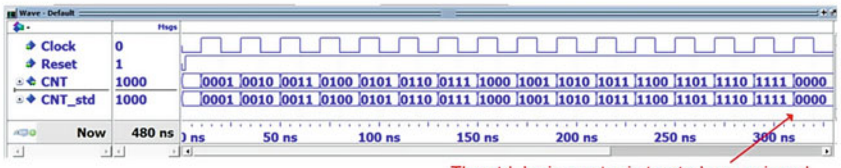
The numeric_std package contains the "+" operator for type integer and a conversion from type integer to type unsigned.

In this example, the output port is defined to be of type unsigned.

An internal signal of type integer is declared to model the counter functionality.

A nested if/then statement checks to see if the integer counter has reached its maximum value.

A concurrent assignment between the internal counter and the output port is made that contains the conversion between type integer and unsigned. The 4 in this function represents the number of unsigned bits to convert the integer into.



The std_logic_vector is treated as unsigned and will roll over once it gets to "1111".

Example 10.2
4-Bit binary up counter in VHDL using the type INTEGER

10.1.3 Counters in VHDL Using the Type STD_LOGIC_VECTOR

It is often desired to have the ports of a system be defined of type std_logic/std_logic_vector for compatibility with other systems. One technique to accomplish this and also model the counter behavior internally using std_logic_vector is through inclusion of the numeric_std_unsigned package. This package allows the use of std_logic_vector when declaring the ports and signals within the design and treats them as unsigned when performing arithmetic and comparison functions. Example 10.3 shows the VHDL model and simulation waveform for this alternative implementation of the 4-bit counter.

Example: 4-Bit Binary Up Counter in VHDL Using the Type STD_LOGIC_VECTOR (1)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.numeric_std_unsigned.all;

entity Counter_4bit_Up is
 port (Clock, Reset : in  std_logic;
       CNT          : out std_logic_vector(3 downto 0));
end entity;

architecture Counter_4bit_Up_arch of Counter_4bit_Up is

 signal CNT_std : std_logic_vector(3 downto 0);

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_std <= "0000";
      elsif (Clock'event and Clock='1') then
        CNT_std <= CNT_std + 1;
      end if;
  end process;

  CNT <= CNT_std;

end architecture;
```

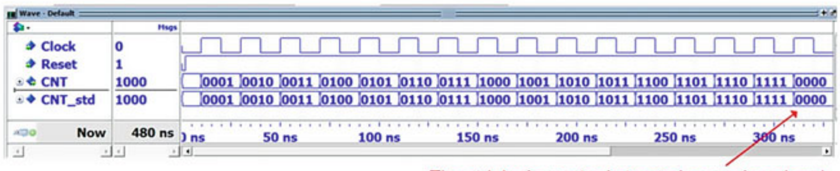
Including this package will treat all std_logic_vector types as unsigned numbers.

The output port is defined to be of type std_logic_vector.

The internal signal to model the counter behavior is declared as type std_logic_vector.

No boundary checking is needed since the 4-bit std_logic_vector will simply roll over.

No type conversion is needed since the internal signal and output port are of type std_logic_vector.



```
Wave - Default
                 Msgs
Clock    0
Reset    1
CNT      1000   0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000
CNT_std  1000   0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000
   Now   480 ns  0 ns   50 ns   100 ns   150 ns   200 ns   250 ns   300 ns
```

The std_logic_vector is treated as unsigned and will roll over once it gets to "1111".

Example 10.3
4-Bit binary up counter in VHDL using the type STD_LOGIC_VECTOR (1)

If it is designed to have an output type of std_logic_vector and use an integer in modeling the behavior of the counter, then a double conversion can be used. In the following example, the counter behavior is modeled using an integer type with range checking. A concurrent signal assignment is used at the end of the architecture in order to convert the integer to type std_logic_vector. This is accomplished by first converting the type integer to unsigned and then converting the type unsigned to std_logic_vector. Example 10.4 shows the VHDL model and simulation waveform for this alternative implementation of the 4-bit counter.

Example: 4-Bit Binary Up Counter in VHDL Using the Type STD_LOGIC_VECTOR (2)

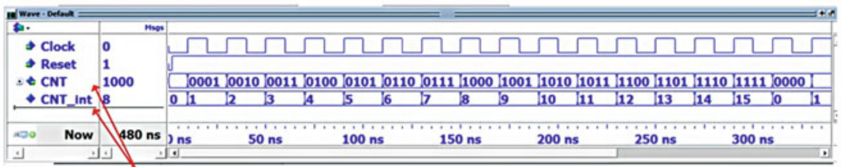
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Counter_4bit_Up is
 port (Clock, Reset : in  std_logic;
       CNT          : out std_logic_vector(3 downto 0));
end entity;

architecture Counter_4bit_Up_arch of Counter_4bit_Up is

 signal CNT_int : integer range 0 to 15;

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_int <= 0;
      elsif (Clock'event and Clock='1') then

          if (CNT_int = 15) then
            CNT_int <= 0;
          else
            CNT_int <= CNT_int + 1;
          end if;

     end if;
  end process;

  CNT <= std_logic_vector( to_unsigned(CNT_int, 4) );

end architecture;
```

The output port is defined to be of type std_logic_vector.

The internal signal to model the counter behavior is declared as type integer. In this declaration, the integer range is also specified. This is unnecessary since the process will check for the maximum counter value but is commonly included for readability.

Range checking is required when using the type integer.

A double type conversion is used to change the integer to std_logic_vector.



In this example, the output CNT is of type std_logic_vector while the counter behavior is modeled using type integer.

Example 10.4
4-Bit binary up counter in VHDL using the type STD_LOGIC_VECTOR (2)

Concept Check

CC10.1 If a counter is modeled using only one process in VHDL, is it still a finite state machine? Why or why not?

(A) Yes. It is just a special case of a FSM that can easily be modeled using one process. Synthesizers will recognize the single process model as a FSM.

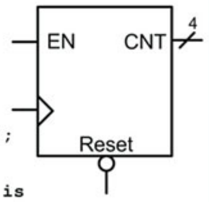
(B) No. Using only one process will synthesize into combinational logic. Without the ability to store a state, it is not a finite state machine.

10.2 Counters with Enables and Loads

10.2.1 Modeling Counters with Enables

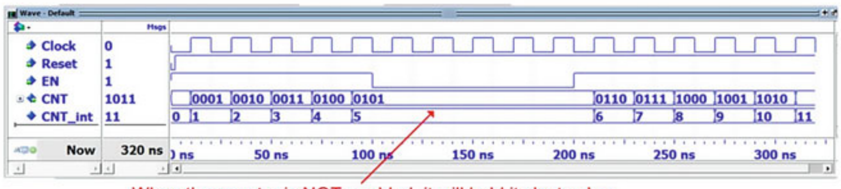
Including an enable in a counter is a common technique to prevent the counter from running continuously. When the enable is asserted, the counter will increment on the rising edge of the clock as usual. When the enable is de-asserted, the counter will simply hold its last value. Enable lines are synchronous, meaning that they are only evaluated on the rising edge of the clock. As such, they are modeled using a nested if/then statement within the if/then statement checking for a rising edge of the clock. Example 10.5 shows an example model for a 4-bit counter with enable.

Example: 4-Bit Binary Up Counter with Enable in VHDL



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Counter_4bit_wEN is
 port (Clock, Reset : in  std_logic;
       EN           : in  std_logic;
       CNT          : out std_logic_vector(3 downto 0));
end entity;

architecture Counter_4bit_wEN_arch of Counter_4bit_wEN is

 signal CNT_int : integer range 0 to 15;

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_int <= 0;
      elsif (Clock'event and Clock='1') then

        if (EN='1') then
            if (CNT_int = 15) then
              CNT_int <= 0;
            else
              CNT_int <= CNT_int + 1;
            end if;
        end if;

     end if;
  end process;

  CNT <= std_logic_vector( to_unsigned(CNT_int, 4) );

end architecture;
```

A nested if/then statement is used in order to check if the counter is enabled on each edge of the clock. If EN='1', the counter will increment as usual. If EN='0', the counter will simply hold its last value.



When the counter is NOT enabled, it will hold its last value.

Example 10.5
4-Bit binary up counter with enable in VHDL

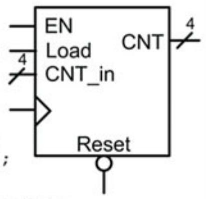
10.2.2 Modeling Counters with Loads

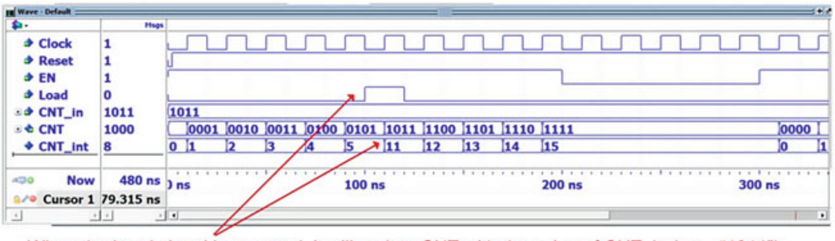
A counter with a *load* has the ability to set the counter to a specified value. The specified value is provided on an input port (i.e., CNT_in) with the same width as the counter output (CNT). A synchronous load input signal (i.e., Load) is used to indicate when the counter should set its value to the value present on CNT_in. Example 10.6 shows an example model for a 4-bit counter with load capability.

Example: 4-Bit Binary Up Counter with Load in VHDL

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Counter_4bit_wLoad is
 port (Clock, Reset : in  std_logic;
       EN           : in  std_logic;
       Load         : in  std_logic;
       CNT_in       : in  std_logic_vector(3 downto 0);
       CNT          : out std_logic_vector(3 downto 0));
end entity;

architecture Counter_4bit_wLoad_arch of Counter_4bit_wLoad is

 signal CNT_int : integer range 0 to 15;

 begin

  COUNTER : process (Clock, Reset)
    begin
      if (Reset = '0') then
        CNT_int <= 0;
      elsif (Clock'event and Clock='1') then

        if (Load = '1') then
           CNT_int <= to_integer( unsigned(CNT_in) );
        else

            if (EN='1') then
                if (CNT_int = 15) then
                  CNT_int <= 0;
                else
                  CNT_int <= CNT_int + 1;
                end if;
            end if;

        end if;

      end if;
  end process;

  CNT <= std_logic_vector( to_unsigned(CNT_int, 4) );

end architecture;
```

A nested if/then statement is used to load CNT with CNT_in when the Load signal is asserted. Since CNT_int is of type integer and CNT_in is of type std_logic_vector, a type conversion is needed. Once again, two conversions are used since there is not a direct conversion between std_logic_vector and integer.





When the Load signal is asserted, it will update CNT with the value of CNT_in (e.g., "1011").

Example 10.6
4-Bit binary up counter with load in VHDL

Concept Check

CC10.2 If the counter has other inputs such as loads and enables, shouldn't they be listed in the sensitivity list along with clock and reset?

(A) Yes. All inputs should go in the sensitivity list.

(B) No. Only signals that trigger an assignment are listed in the sensitivity list. The only two signals that have this behavior are clock and reset.

Summary

- A counter is a special type of finite state machine in which the states are traversed linearly. The linear progression of states allows the next state logic to be simplified. The complexity of the output logic in a counter can also be reduced by encoding the states with the desired counter output for that state. This technique, known as *state-encoded outputs*, allows the system outputs to simply be the current state of the FSM.
- Counters are a special type of finite state machine that can be modeled using a single process in VHDL. Only the clock and reset signals are listed in the sensitivity list of the counter process because they are the only signals that trigger signal assignments.
- Within the process of a counter, arithmetic operators (i.e., + or –) can be used to modify the counter value. Since these operators aren't defined for the type std_logic_vector, type casting is usually required.

Exercise Problems

Section 10.1: Modeling Counters with a Single Process

10.1.1 Design a VHDL behavioral model for a 16-bit, binary up counter using a single process. The block diagram for the entity definition is shown in Fig. 10.1. In your model, declare Count_Out to be of type unsigned, and implement the internal counter functionality with a signal of type unsigned.

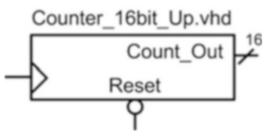


Fig. 10.1
16-Bit binary up counter block diagram

10.1.2 Design a VHDL behavioral model for a 16-bit, binary up counter using a single process. The block diagram for the entity definition is shown in Fig. 10.1. In your model, declare Count_Out to be of type unsigned and implement the internal counter functionality with a signal of type integer.

10.1.3 Design a VHDL behavioral model for a 16-bit, binary up counter using a single process. The block diagram for the entity definition is shown in Fig. 10.1. In your model, declare Count_Out to be of type std_logic_vector and implement the internal counter functionality with a signal of type integer.

Section 10.2: Counters with Enables and Loads

10.2.1 Design a VHDL behavioral model for a 16-bit, binary up counter with enable using a single process. The block diagram for the entity definition is shown in Fig. 10.2. In your model, declare Count_Out to be of type unsigned and implement the internal counter functionality with a signal of type unsigned.

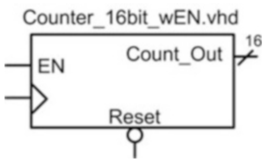


Fig. 10.2
16-Bit binary up counter with enable block diagram

10.2.2 Design a VHDL behavioral model for a 16-bit, binary up counter with enable using a single process. The block diagram for the entity definition is shown in Fig. 10.2. In your model, declare Count_Out to be of type unsigned, and implement the internal counter functionality with a signal of type integer.

10.2.3 Design a VHDL behavioral model for a 16-bit, binary up counter with enable using a single process. The block diagram for the entity definition is shown in Fig. 10.2. In your model, declare Count_Out to be of type std_logic_vector, and implement the internal counter functionality with a signal of type integer.

10.2.4 Design a VHDL behavioral model for a 16-bit, binary up counter with enable and load using a single process. The block diagram for the entity definition is shown in Fig. 10.3. In your model, declare Count_Out to be of type unsigned, and implement the internal counter functionality with a signal of type unsigned.

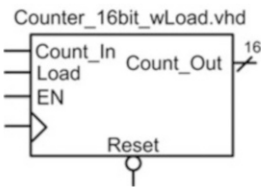


Fig. 10.3
16-Bit binary up counter with load block diagram

10.2.5 Design a VHDL behavioral model for a 16-bit, binary up counter with enable and load using a single process. The block diagram for the entity definition is shown in Fig. 10.3. In your model, declare Count_Out to be of type unsigned, and implement the internal counter functionality with a signal of type integer.

10.2.6 Design a VHDL behavioral model for a 16-bit, binary up counter with enable and load using a single process. The block diagram for the entity definition is shown in Fig. 10.3. In your model, declare Count_Out to be of type std_logic_vector, and implement the internal counter functionality with a signal of type integer.

10.2.7 Design a VHDL behavioral model for a 16-bit, binary up/down counter using a single process. The block diagram for the entity definition is shown in Fig. 10.4. When Up = 1, the counter will increment. When Up = 0, the counter will decrement. In your model, declare Count_Out to be of type unsigned, and implement the internal counter functionality with a signal of type unsigned.

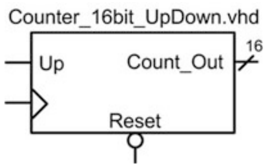


Fig. 10.4
16-Bit binary up/down counter block diagram

10.2.8 Design a VHDL behavioral model for a 16-bit, binary up/down counter using a single process. The block diagram for the entity definition is shown in Fig. 10.4. When Up = 1, the counter will increment. When Up = 0, the counter will decrement. In your model, declare Count_Out to be of type unsigned and implement the internal counter functionality with a signal of type integer.

10.2.9 Design a VHDL behavioral model for a 16-bit, binary up/down counter using a single process. The block diagram for the entity definition is shown in Fig. 10.4. When Up = 1, the counter will increment. When Up = 0, the counter will decrement. In your model, declare Count_Out to be of type std_logic_vector, and implement the internal counter functionality with a signal of type integer.