SECTION 4

COMPUTER-ASSISTED DIGITAL SYSTEM DESIGN

Computer-assisted design continues to become a more powerful tool in digital and analog circuit design, especially using programmable integrated circuits. The different types and manufacturers of such devices fall into two main classes—field programmable gate arrays (FPGAs) and field programmable analog arrays (FPAAs).

We will focus on digital circuits (FPGAs) using the most general process of design. However, the approach works well on all of the rest of the commercial CAD tools and can be applied to the analog devices (FPAAs) as well. C.A.

In This Section:

CHAPTER 4.1
DESIGN OF DIGITAL SYSTEMS USING CAD TOOLS

Brian R. Fast

OVERVIEW

The use of CAD tools for programming digital circuits has dramatically increased the reusability, processing power, and reliability of digitally designed components. Currently field programmable gate array (FPGA) chips use very high-speed integrated circuits hardware description language (VHDL) to take advantage of these CAD tools in the programming of chips at the lowest gate level. VHDL has become an industry standard language, which is regulated by the Institute of Electrical and Electronic Engineers (IEEE). It is similar, with many conceptual changes described later in this section, to any other programming language (e.g., C, C++, visual basic). The companies that manufacture FPGA devices have software packages that can compile VHDL code that can be simulated or burned to a chip.

One of the potential benefits to VHDL over chip-depended software is the capability to migrate from one chip manufacturer to another. VHDL is an industrial standard language, therefore the software for each chip manufacture should be capable of compiling code designed for another manufacturer's chip. Of course this is dependent on the code not using functions that are proprietary to a specific chip manufacturer.

The increase in processing power is another major advantage of FPGA devices over the standard microprocessor. There are two avenues in which this can be seen. First, the capability of performing concurrent operations is a major advantage of FPGA devices. One chip can be broken into subsystems that perform the processes in parallel. If this was to be implemented with microprocessors there would need to be an independent processor for each subsystem. This would considerably increase the overall design cost and add complexity to system integration. Second, the capability of designing the number of busses to the desired size increases the efficiency and optimizes the design. For example, standard microprocessors have a fixed number of busses and register sizes. FPGA devices are programmed at the gate level, which gives the designer the added freedom to optimize the size and the number of registers throughout the design.

The added reliability of the design can be seen through improved CAD tools and the ease of reusability. CAD tools now provide powerful simulation packages that enable the designer to thoroughly test the components. These simulation packages are typically not as usable in microprocessor packages. For example, the simulation packages for FPGA devices produce waveforms that can be included with testing documentation. Microprocessor-based design can be simulated to show the desired input-output response; however, all of the intermediate details are typically negated. The ease of reusability also increases the reliability of the design, as the component can be used again and again, the confidence level in the component improves. This is also easier to do in an FPGA-based design over a microprocessor design, because integration of the designed component is easier in an FPGA-based design.

There are, however, some disadvantages to the more powerful FPGA-based chip design. Designing with VHDL requires the designer to think in terms of hardware instead of software. This is a major change in programming

philosophy, which will take some time for the new designer to become accustomed to. It requires the designer to think of the design as a high-level system, and as signals moving throughout the system. The designer must design all of the overhead tasks that are required (internal interrupts, external interrupts, watchdog timers). The FPGA-based chips do not typically have preconfigured devices on the chip (analog-to-digital converter, pulse-width modulator, communication routines). However, the newer devices are beginning to add a standard microprocessor within FPGA device, which would ease some of the implementation burdens of standard tasks.

ALTERNATIVE PROGRAMMING PHILOSOPHY (BASIC)

One of the major difficulties in working with VHDL/FPGA devices is the alternative programming philosophy. Typically programmers are used to thinking in a linear fashion where variables are thought to contain specific information. The information that is held at a specific moment is dependent on where the program is operating. VHDL requires the programmer to change their method of thinking to become a designer. It requires the designer to think of the variables as wires, which move signals throughout the design rather than moving information. Let us take a look at the design philosophy for a microprocessor and compare it to the VHDL design for a simple problem shown in Eq. (1).

$$F <= (a \text{ AND } b) \text{ AND } (c \text{ AND } d) \tag{1}$$

A typical microprocessor design would begin by loading the contents of *a* to *a* register, loading the contents of *b* to another register, and the two registers together, and then store that value in a designated area 1. It would then load the contents of *c* to *a* register, load the contents of *b* to another register, and the two registers together, and then store that value in a designated area 2. It would then load the contents of the designated area 1 to a register, load the contents of the designated area 2 to the register, and the two registers together, and then there is the output. The program would then be repeated over and over to update the changes as *a*, *b*, *c*, or *d* change. From this process it is seen that programming of a microprocessor occurs, like many programming languages, linearly throughout the appropriate steps.

The implementation of Eq. (1) in VHDL is rather straightforward. However, with this comparison, the difference in the design process is very apparent. For now, the implementation of the VHDL code will be done in a concurrent manner. Later on, the design can be modified to simulate a linear operation. The VHDL designer would first decide if this function is the minimum design needed to generate the appropriate output. This function is of the minimum design, therefore the function will be directly implemented in VHDL. Basically the circuit requires that there be three AND gates. The signals *a* and *b* are input to one of the AND gates. The signals *c* and *d* are input to the second AND gate. The output of the first and second AND gates are run to the third AND gate that provides the final result. The specific syntax of VHDL will be discussed in the subsequent sections; however, Eq. (1) is also the VHDL code for the desired function. Basically, when sequential statements are not used, in VHDL code the output will change immediately (within a small time delay) following a change in *a*, *b*, *c*, or *d*. The code would not need a repeating loop to update the output of the function as the values of *a*, *b*, *c*, or *d* change. The VHDL code would provide the result in an order of magnitude faster than the microprocessor design, and it would also be capable of performing other operations simultaneously. The change from a linear programming to concurrent programming sounds easy; however, it takes a little more effort than would be anticipated.

The most important part of programming in this type of environment is having a good picture of the high-level objective. Having a good high-level diagram is a must for designing VHDL code. An example of the high-level code for Eq. (1) is shown in Fig. 4.1.1. The high-level diagram shown in Fig. 4.1.1 is rather simple for this problem, but as systems become more complicated with multiplexers and other functions, the high-level diagram will make it possible to keep track of what each signal is. This will be seen in the alternative programming section as the high-level diagrams become more complicated, which requires sequential and concurrent processing.

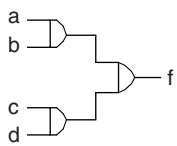

FIGURE 4.1.1 High-level diagram for Eq. (1).

```
-- ************************************************************
-- Include Libraries
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE ieee.std_logic_unsigned.ALL;
-- ************************************************************
```

FIGURE 4.1.2 VHDL code for include libraries.

VHDL/FPGA Program Configuration

This will be the first formal introduction to the syntax of VHDL. There are three major components to the syntax. There are the included libraries, there is the entity that defines the inputs and outputs of the component, and there is the architecture that is the main body of the code.

The included libraries provide standard functions, which are available within the architecture. While deciding which libraries to include, one may want to consider what other chip manufactures support these libraries, in case in the future, a decision is made to use another chip manufacture. The syntax for the libraries is shown in Fig. 4.1.2. Note that two dashes (--) indicates a comment.

The entity section tells the system what inputs and outputs will be coming into and out of the component and how they will be defined. A comparison to a standard programming language would be the type of variable (int, double, float). The syntax for the entity is shown in Fig. 4.1.3.

The main purpose of the entity declaration is to define the signals as input or output and define the size of the signals. For example, number is defined as a vector that is 8 bits wide. In the CAD software the input and output signals will be referenced to a specific I/O port. Take note that an output cannot be feedback to a defined input signal. An input must be set by a signal external to the device. The input and output signals that are declared as STD_LOGIC means that these are only a single bit.

The architecture section defines the actual function of the chip. Within the architecture, there are a few more important declarations. These declarations are known as signals (intermediate locations to store signals), constants, and variables. Variables are only allowed to be used within a sequential statement, which will be covered in subsequent sections. The syntax of the architecture is shown in Fig. 4.1.4.

VHDL/FPGA Concurrent Programming

Very high-speed integrated circuits hardware description language is capable of performing concurrent and sequential operations. This section will focus on concurrent processing; in a later section sequential operations will be introduced. The concurrent processing methodology is the main difference from microprocessor-type design. Concurrent processing means that if one of the inputs changes its value then the output will, nearly, immediately change as well. There isn't a sense of looping or linear programming. Essentially the entire program acts in a parallel manner.

```
-- ********************************************************************************
-- Declaration of INPUT & OUTPUT variables
-- ENTITY must have a reference, which is the same as  the file name
ENTITY example IS
       PORT(          clk            : IN   STD_LOGIC;
                      button         : IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
                      flag           : OUT  STD_LOGIC;
                      number         : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0));
END example;
-- ********************************************************************************
```

FIGURE 4.1.3 VHDL code for entity.

```
-- ********************************************************************************
-- The actual body of the program
ARCHITECTURE arch OF example IS
        -- Declare constant as needed
        CONSTANT       freq :                integer := 12587500;
        -- Declare signals and type as needed
        SIGNAL  number_now :   std_logic_vector(7 DOWNTO 0);
        SIGNAL  number_next:   std_logic_vector(7 DOWNTO 0);
        -- Declare variables
        BEGIN
        -- start of actual body of program
                -- input desired code here
        -- end of the actual body of program
END arch;
-- ********************************************************************************
```

FIGURE 4.1.4 VHDL code for architecture.

Within this section the designer has access to multiplexers, basic bit operations (AND, OR, NOT, and so forth), and some other general commands. A few of the multiplexer techniques are the WITH and WHEN statement and will be shown in Figs. 4.1.5 and 4.1.6.

In Fig. 4.1.5 the WITH statement is implemented. The actual purpose of the program is not important. It is only shown to illustrate the syntax. Within the figure there is another command that is implemented and that is the "&." The & command enables the designer to append static or dynamic signals within another signal. Another application of the & is a shift operation.

In Fig. 4.1.6 the WHEN statement is implemented. The actual purpose of the program is not important. It is only shown to illustrate the syntax. Another important aspect to all of the multiplexer commands is to remember to include all possible combinations of the select signal. If all possible combinations are not represented, undesirable results could occur.

ALTERNATIVE PROGRAMMING PHILOSOPHY (ADVANCED)

One of the major difficulties of working with FPGA devices is having to design circuits in a concurrent manner. This section will present a method of removing that limitation. There is a method of designing sequential circuits, essentially creating a circuit that functions in a linear fashion. This type of program is called a *process(sel)*. The process function also has a list of signals that are evaluated by the process. There are no fixed number of signals that can be evaluated by the process. The process will be ignored until the *sel* signal changes. Once the signal changes, the process will be entered and the appropriate code will be evaluated.

```
-- ****************************************************************
-- The actual body of the program
-- This routine is the meat of the frequency divider
ARCHITECTURE arch OF example IS
BEGIN
     -- shows how to use a WITH statement
     WITH button SELECT
       number <= "1000000" & "0"     WHEN "1000",
                 "010000" & "00"     WHEN "0100",
                 "00100" & "000"     WHEN "0010",
                 "0001" & "0000"     WHEN "0001",
                 "00000000"          WHEN OTHERS;
END arch;
-- ****************************************************************
```

FIGURE 4.1.5 VHDL code for WITH.

```
-- ************************************************************************
-- The actual body of the program
-- This routine is the meat of the frequency divider
ARCHITECTURE arch OF example IS
BEGIN
     -- shows how to use a WHEN statement
     number <= "100000000"    WHEN (button = "1000") ELSE
               "010000000"    WHEN (button = "0100") ELSE
               "001000000"    WHEN (button = "0010") ELSE
               "000100000"    WHEN (button = "0001") ELSE
               "000000000";
END arch;
-- ************************************************************************
```

FIGURE 4.1.6 VHDL code for WHEN.

When designing with a process the designer gains access to a new type of command. This new command is a variable. A variable can only be accessed within a process. This new type of command is needed because of the limitation of working with signals within a process. Within a process the signal will not be updated until the process is left, which means if the signal is changed many times within one process, only the last value of the signal will take effect. The change in the signal will not take place until the process is left. The use of a variable creates difficulty because of the lack of knowledge of how the gate-level design will be configured. This causes the size of the design to grow larger than the use of a typical signal. The added advantage of using a variable will allow the designer to make changes to the variable within the process many times before the process is completed.

There are many advantages to designing a sequential circuit. The most obvious is the capability of taking advantage of a looping statement and an "if" statement, however, the most important addition would be the way the high-level design could be reformulated to create a finite state machine. Essentially being able to break a task into multiple states will enable the designer to systematically solve the problem. During the implementation of a finite state machine a state and a next state need to be known at all times. Another key importance of this type of design is the ability to perform multiple operations on a signal. Remember typically a signal cannot write to itself. However, using a current and next state vector for the signal will enable the designer to get around this limitation. The high-level diagram is shown in Fig. 4.1.7.

Figure 4.1.7 shows how to visualize the state update components. The system begins with two separate processes that depend on two different signals. The first process is dependent on a change in the state_reg register. The second process is dependent on a change in the clock. The first process keeps track of the current state and monitors other signals, not shown, to determine when and what state to post to the state_next register. The second process takes into account a clock signal, which waits for the rising edge of the clock to update the state_reg with the state_next register.

The configuration of the state update component is used to streamline the design in order to reduce the amount of spaghetti code. This configuration also provides a method of performing multiple instances of a function to a signal.

There are a few major drawbacks to this type of design. First, when designing a circuit sequentially the way the circuit is compiled at the gate-level becomes less obvious. This causes the circuit to take up more space within the chip. Second, sequential design runs the possibility of not being able to compile even if the syntax is correct. The resulting sequential design might be too complex.

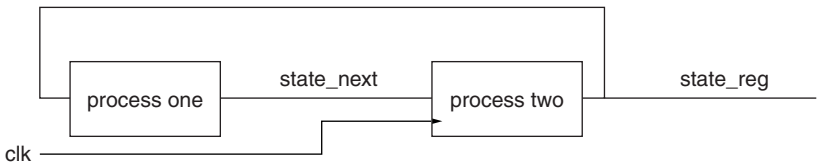

FIGURE 4.1.7 High-level diagram of state update philosophy.

```
-- ********************************************************************************************
-- The actual body of the program
-- This routine is the meat of the frequency divider
ARCHITECTURE arch OF clkdivider IS
     SIGNAL      count_now:    std_logic_vector(27 DOWNTO 0);
     SIGNAL      count_next:   std_logic_vector(27 DOWNTO 0);
BEGIN
     PROCESS(clk)
        BEGIN
        -- increments count_next each time the input signal
        -- goes high
        if(clk ='1' AND clk'EVENT) THEN
                count_next <= count_now +"0000000000000000000000000001";
        END IF;
     END PROCESS;
     -- resets the counter once the system has gone through the
     -- number of cycles
     PROCESS(count_next)
        BEGIN
        IF(count_next = "1111000000000000000000000000") THEN
             count_now <= "0000000000000000000000000000";
        ELSE
             count_now <= count_next;
        END IF;
     END PROCESS;
END arch;
-- ********************************************************************************************
```

FIGURE 4.1.8 VHDL code for a process and performing multiple instances of a function on a signal.

VHDL/FPGA Sequential Programming

This section will focus on presenting a sequential programming syntax for VHDL. The main illustrations will show how to set up a process, read the rising edge of the clock signal, and how to perform multiple instances of a function on one signal.

Figure 4.1.8 shows the typical syntax for a process and how to determine the rising edge of the clk signal. The implementation of the architecture shown in Fig. 4.1.8 also shows how to perform multiple instances of a function to a signal. This method is done by creating a current signal (defined as count_now) and by creating a next signal (defined as count_next).

Creating a process to implement the state update component shown in Fig. 4.1.7 would follow the same methodology of the process shown in Fig. 4.1.8. Instead of updating the next state by evaluating a function, the output would be the next state, which is dependent on the current state and the inputs to the component.

Timing Considerations

Timing considerations are the most critical components to any embedded system design. This is true with FPGA-based designs as well. The requirements that are standard to any embedded system still remain; however, there is a new timing requirement that must be considered in order to optimize the system. In the development of the states of the system, the designer must consider whether to have fewer states with more packed into the function or to have more states with less packed into the function. The key trade-off is throughput versus clock speed. Within the process that updates the current state based on the clock frequency, the max clock frequency would be dependent on the longest cycle through the process. Therefore, it is typically more desirable to have more states that implement less than to have less states that implement more. The timing considerations and max clock frequency can be determined using the CAD tools.

Simulation with CAD Tools

A key advantage to using the new CAD tools is the powerful set of simulation packages. Windows can be created, which show the waveform for any signal throughout the system. The simulations can show the final value and any intermediate value throughout the design. This is a major advantage for debugging and system verification.

VHDL/FPGA Component Integration and Configuration

Integrating many VHDL components together into a big design is much easier than implementing functions within a typical microprocessor. The overall design can be broken into components and each component can be independently designed and compiled. The final product can link the components together to form the overall design.

In order to illustrate this concept, a kitchen timer example will be shown. The actual function of the design is not as important as how the system is integrated together. However, for clarity, a brief explanation will be given. The system begins by placing the initial value to begin counting from LSreg (seconds) and MSreg (tens of seconds). There will be a clock divider, which will create a 1-s pulse for the counter circuit to count in order to update the clock every second. The counter component will take the initial values of LSreg and MSreg and decrement the value every second until the timer reaches zero. The updated counter will be output to the seven seg display component, which will display the appropriate number on the display. This process will stop when the counter reaches zero or it will reset when the Reset_Switch is initiated. The high-level design is shown in Fig. 4.1.9.

Within Fig. 4.1.9 it is seen that there are four major subsystems for the kitchen timer example—complete system, clkdivider, counter, and the seven seg display. Each of these subsystems is an individual component, individual file, in the design of the overall system. The inputs and outputs of each block are declared and the function of each block is declared within the architecture. The clkdivider, counter, and seven seg display are the core functions, which dictate what is actually happening in the circuit. The complete system component links all of the subsystems together.

Within the complete system component the design will define the inputs and outputs to the entire system and link the inputs and outputs of each component together. There is one more function of the complete system component, which will complete the design. When there is a signal, which is an output from one component and an input to another component (not an output for the system), there needs to be an intermediate signal defined, which will be used to link the signal. Figure 4.1.10 shows the code that was designed for the complete system component.

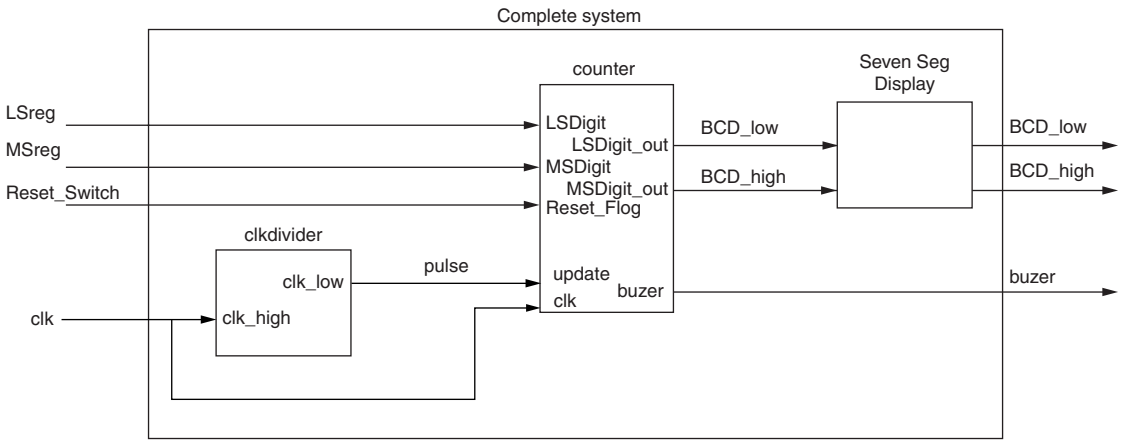

FIGURE 4.1.9 High-level diagram of kitchen timer example.

```
-- ************************************************************
-- Program Name     :    completesystem.vhd
-- File Name        :    desktop\new folder\altera\lab 2b
-- Programed By     :    Brian Fast
-- Date             :    April 1, 2004
-- Purpose          :    This Program will pull all of the subsystems
--                       together for the kitchen timer program
--                       1) clock divider
--                       (output a rising pulse every second)
--                       2) Binary To Decimal Decoder and Subtractor
--                       3) 7-Seg Display
-- Input    :     clock signal -> 25.175 MHz
--                Reset_Switch -> pull high then low to start
--                LSreg -> binary representation of decimal value
--                input valid only between 0-9 but can handle
--                15-0 but anything above 9 will default to zero
--                MSreg -> binary representation of decimal value
--                input valid only between 0-9 but can handle
--                15-0 but anything above 9 will default to zero
-- Output   :     LS7seg -> seven segment display decimal representation of
--                Least Significant Digit of counter
--                MS7seg -> seven segment display decimal representation of
--                Most Significant Digit of Counter
-- Chip     :     10K70RC240-4
-- Board    :     Altera University Program Development Board UP2
--
-- NOTE     :     Must Compile These Files and have them
--                in the same folder
--                1) counter.vhd
--                2) clkdivider.vhd
--                3) sevensegdisplay.vhd
--
--                This program creates a program that joins all
--                of the subprograms compiled before
--
--                I/O pin numbers are can be predefined in the
--                .acf file
--                I/O pin number verification be seen in the
--                .rpt file
--
-- ************************************************************

-- Include Libraries
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
-- ************************************************************
-- Declaration of INPUT & OUTPUT variables
-- ENTITY must have a reference which is the same as  the file name

ENTITY completesystem IS
        PORT(   clk                     :       IN      std_logic;
                LSreg                   :       IN      std_logic_vector(3 DOWNTO 0);
                MSreg                   :       IN      std_logic_vector(3 DOWNTO 0);
                Reset_Switch            :       IN      std_logic;
                LS7seg                  :       OUT     std_logic_vector(6 DOWNTO 0);
                MS7seg                  :       OUT     std_logic_vector(6 DOWNTO 0);
                DP7seg                  :       OUT     std_logic_vector(1 DOWNTO 0);
                Buzer                   :       OUT     std_logic);
END completesystem;
-- ************************************************************
```

FIGURE 4.1.10 VHDL code for the complete system component that links all the other components together.

```
-- ************************************************************
-- The actual body of the program
-ARCHITECTURE arch OF completesystem IS
        -- component declaration
        -- files which are interconnected to make up the entire file
        -- include:
        --              file name
        --              inputs
        --              outputs
        COMPONENT clkdivider
                PORT(
                        clk_high        :       IN      STD_LOGIC;
                        clk_low         :       OUT     STD_LOGIC);
        END COMPONENT;

        COMPONENT counter
                PORT(
                        clk             :       IN      STD_LOGIC;
                        update          :       IN      STD_LOGIC;
                        LSDigit         :       IN      STD_LOGIC_VECTOR(3 DOWNTO 0);
                        MSDigit         :       IN      STD_LOGIC_VECTOR(3 DOWNTO 0);
                        Reset_Flag      :       IN      STD_LOGIC;
                        LSDigit_out     :       OUT     STD_LOGIC_VECTOR(3 DOWNTO 0);
                        MSDigit_out     :       OUT     STD_LOGIC_VECTOR(3 DOWNTO 0);
                        Buzer           :       OUT     STD_LOGIC);
        END COMPONENT;

        COMPONENT sevensegdisplay
                PORT(
                        LSinput         :       IN      STD_LOGIC_VECTOR(3 DOWNTO 0);
                        MSinput         :       IN      STD_LOGIC_VECTOR(3 DOWNTO 0);
                        LSdisplay       :       OUT     STD_LOGIC_VECTOR(6 DOWNTO 0);
                        MSdisplay       :       OUT     STD_LOGIC_VECTOR(6 DOWNTO 0);
                        DPdisplay       :       OUT     STD_LOGIC_VECTOR(1 DOWNTO 0));
        END COMPONENT;

        -- interconnection signals
        -- signals that connect outputs to inputs within the system
        SIGNAL pulse    : STD_LOGIC;
        SIGNAL BCD_low  : STD_LOGIC_VECTOR(3 DOWNTO 0);
        SIGNAL BCD_high : STD_LOGIC_VECTOR(3 DOWNTO 0);

        BEGIN
        -- mapping the inputs and outputs to and from each subsystem
                clkdivider_unit: clkdivider
                PORT MAP(clk_high=>clk, clk_low=>pulse);

                counter_unit: counter
                PORT MAP(clk=>clk, update=>pulse, Reset_Flag=>Reset_Switch, LSDigit=>LSreg,
                        MSDigit=>MSreg, LSDigit_out=>BCD_low, MSDigit_out=>BCD_high,
                        Buzer=>Buzer);

                sevensegdisplay_unit: sevensegdisplay
                PORT MAP(LSinput=>BCD_low, MSinput=>BCD_high, LSdisplay=>LS7seg,
                                MSdisplay=>MS7seg, DPdisplay=>DP7seg);

END arch;
-- ************************************************************
```

FIGURE 4.1.10 (*Continued*) VHDL code for the complete system component that links all the other components together.

EXAMPLE DESIGN

A complete example design will be shown in this section. The example will be the remaining components of the kitchen timer problem. The complete system code has already been shown in Fig. 4.1.10. The remainder of the components shall be shown in Figs. 4.1.11 to 4.1.13.

```
-- ****************************************************************
-- Program Name      :     clkdivider.vhd
-- File Name         :     desktop\new folder\altera\kitchen timer
-- Programed By      :     Brian Fast
-- Date              :     April 2, 2004
-- Purpose           :     Clock Divider
--                       take a high frequency clock and output a
--                       lower frequency cycle
-- Input       :         currently configured for system clock (25.175 MHz)
--                       can be changed by changing the constant values
--                       freq_in = input frequency/(2*desired output frequency)
--                       freq_in_switch = (input frequency/(2*desired output frequency))/2
-- Ouput       :         currently configured for clock signal at (60 Hz)
-- Chip        :         10K70RC240-4
-- Board       :         Altera University Program Development Board UP2
-- Software    :         Altera Max+Plus v10.22
--
-- NOTE        :         This file will be used as a black box within
--                       another file so the I/O pins will not be
--                       set within this file.  The I/O signals will
--                       be interconnections set within the main program
--                       which is where the I/O pins will be set
--
--              Num      Description                                    By      Date
-----------------------------------------------------------------------------------------------------------------
-- Status    :  1.0      Began Initial Program                          BRF     4/2/2004
--                       The program is working correctly
--                       The program takes an input frequency
--                       and outputs a lower frequency
--                       which is dependent on the values set
--                       in the constants
--                       freq_in & freq_in_switch
--                       these constants are dependent on the
--                       input frequency and the desired
--                       output frequency
--                       the input output frequency is not
--                       determined but is dependent on the
--                       input frequency output frequency
--                       size of the registers and the
--                       internal speed of the code
--
--                                    ------------
--                                   | freq       |
--              high frequency ----->| divider    |---> lower frequency
--              system clock         |            |     new desire frequency
--              [clk_high]            ------------      [clk_low]
--
--
-- ****************************************************************
```

FIGURE 4.1.11 VHDL code for the clock divider component.

```
-- Include Libraries
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE ieee.std_logic_unsigned.ALL;
-- ************************************************************
-- Declaration of INPUT & OUTPUT variables
-- ENTITY must have a reference which is the same as  the file name

ENTITY clkdivider IS
       PORT(   clk_high              : IN    STD_LOGIC;
               clk_low               : OUT   STD_LOGIC);
END clkdivider;
-- ************************************************************
-- ************************************************************
-- The actual body of the program
-- This routine is the meat of the frequency divider

ARCHITECTURE arch OF clkdivider IS
-- Set constant by the equation below
-- freq_in = (the input frequency/(2*desired frequency))
-- the max freq_in is dependent on the output frequency
CONSTANT       freq_in        :       integer := 12587500;

-- Set constant by the equation below
-- freq_in_switch = (the input frequency/desired frequency)/2
CONSTANT       freq_in_switch :       integer := 6293750;

-- used for testing
--CONSTANT     freq_in        :       integer := 34215;
--CONSTANT     freq_in_switch :       integer := 17107;
-- temporary registers used to keep track of how many input signals
-- have been input to the system
SIGNAL         count_now:             std_logic_vector(27 DOWNTO 0);
SIGNAL         count_next:            std_logic_vector(27 DOWNTO 0);

BEGIN
       PROCESS(clk_high)
              BEGIN
              -- increments count_next each time the input signal
              -- goes high
              -- keeps tracked of the number of cycles input by the system clock
              -- via clk_high
              if(clk_high ='1' AND clk_high'EVENT) THEN
                           count_next <= count_now
                           +"0000000000000000000000000001";
              END IF;
              -- determines if the output signal should be high or low
              -- by waiting until based on the constants set above
              -- ie. the input frequency and the desire output frequency
              IF(count_now < freq_in_switch) THEN
                     clk_low <= '1';
              ELSE
                     clk_low <= '0';
              END IF;
       END PROCESS;
       -- resets the counter once the system has gone through the
```

FIGURE 4.1.11 (*Continued*) VHDL code for the clock divider component.

```
        -- number of cycles equal to the input frequency
        PROCESS(count_next)
                BEGIN
                IF(count_next = freq_in) THEN
                        count_now <= "00000000000000000000000000";
                ELSE
                        count_now <= count_next;
                END IF;
        END PROCESS;
END arch;
-- ***************************************************************
-- END OF PROGRAM
```

FIGURE 4.1.11 (*Continued*) VHDL code for the clock divider component.

```
-- ***************************************************************
-- Program Name        :       counter.vhd
-- File Name           :       desktop\new folder\altera\kitchen timer
-- Programed By        :       Brian Fast
-- Date                :       April 3, 2004
-- Purpose             :       BCD Counter
--           The program will take the 4 bit binary number input to the system
--           on the MSDigit and LSDigit and initialize the counter
--           to that value then the reset is pulled high then low.
--           The inputs are 2 registers - 4 bit which represent a decimal value
--           one for the one's place holder and one for the ten's place holder
--           The system will then decrement the value
--           by one every time that the update pin has a positive
--           transition.  The decremented value will then be
--           put on the output pins.
-- Input    :   LSDigit = connected to four input pins = input binary number on pins
--              MSDigit = connected to four input pins = input binary number on pins
--              the inputs can handle a decimal number from 15-0
--              only an entry between 9-0 will be accepted
--              any number greater then 9 will default back to 9
--              this limitation is due to the fact that the counter is intended
--              to count in decimal with decimal inputs
-- Ouput    :   LSDigit_next
--              MSDigit_next
--              outputs can be decimal number from 15-0 but only an output of
--              the range 0-9 decimal will be output
-- Chip     :   10K70RC240-4
-- Board    :   Altera University Program Development Board UP2
-- Software :   Altera Max+Plus v10.22
--
-- NOTE     :   This file will be used as a black box within
--              another file so the I/O pins will not be
--              set within this file.  The I/O signals will
--              be interconnections set within the main program
--              which is where the I/O pins will be set
--
--              Num     Description                             By              Date
-------------------------------------------------------------------------------------
-- Status :     1.0     Began Initial Program                   BRF             4/2/2004
--              1.01    Program Working for BCD Counter         BRF             4/6/2004
```

FIGURE 4.1.12 VHDL code for the counter component.

```
--
--                                        -------------
--                                       |             |
--      system clock [clk]  (1 bit ) ---->|    2 seg    |---> [LSDigit_out]  (4 bits)
--                 [update] (1 bit ) ---->|     BCD     |---> [MSDigit_out]  (4 bits)
--               [LSDigit]  (4 bits) ---->|   counter   |
--               [MSDigit]  (4 bits) ---->|            0|---> [Buzer]  (1 bit)
--            [Reset_Flag]  (1  bit) ---->|             |
--                                        -------------
--                                    Currently this program is decrementing a 2 seg
--                                    BCD counter from the value loaded down to
--                                    zero. The program begins when the Reset_Flag
--                                    is pulled high and then low. The program
--                                    continues to decrement the value until it
--                                    reaches zero zero. Then the value remains
--                                    zero zero until a new value is loaded.
--
-- ***************************************************************
-- Include Libraries
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE ieee.std_logic_unsigned.ALL;

-- ***************************************************************
-- Declaration of INPUT & OUTPUT variables
-- ENTITY must have a reference which is the same as the file name

ENTITY counter IS
        PORT(           clk             : IN    STD_LOGIC;
                        update          : IN    STD_LOGIC;
                        LSDigit         : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
                        MSDigit         : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
                        Reset_Flag      : IN    STD_LOGIC;
                        LSDigit_out     : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0);
                        MSDigit_out     : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0);
                        Buzer           : OUT   STD_LOGIC);
END counter;
-- ***************************************************************
-- ***************************************************************
-- The actual body of the program
-- This routine is the meat of the frequency divider

ARCHITECTURE arch OF counter IS
SIGNAL    LSDreg_now    :       std_logic_vector(3 DOWNTO 0);
SIGNAL    LSDreg_next   :       std_logic_vector(3 DOWNTO 0);
SIGNAL    MSDreg_now    :       std_logic_vector(3 DOWNTO 0);
SIGNAL    MSDreg_next   :       std_logic_vector(3 DOWNTO 0);
SIGNAL    Done_Flag     :       std_logic;
BEGIN
        ---------------------------------------------------------------
        -- Counter Routine
        -- counts down the Least Significant Digit until it gets to zero
        -- then decrements the Most Significant Digit
        -- then decrements resets Least Significant Digit back to nine
        -- until it gets to zero zero
```

FIGURE 4.1.12 (*Continued*) VHDL code for the counter component.

```
-- then the buzzer goes off
-- if the initial value set for the input is greater then 9
-- then the initial value will be set to 9
PROCESS(clk)
BEGIN
        IF(clk = '1' AND clk'EVENT) THEN
                -- checks to see if the value is greater then
                IF(MSDreg_now > 9 OR LSDreg_now > 9) THEN

                        IF(MSDreg_now > 9) THEN
                                MSDreg_next <= "1001";
                        END IF;

                        IF(LSDreg_now > 9) THEN
                                LSDreg_next <= "1001";
                        END IF;

                ELSIF(Done_Flag = '0') THEN
                        IF(LSDreg_now > 0) THEN
                                LSDreg_next <= LSDreg_now - 1;
                                MSDreg_next <= MSDreg_now;
                        ELSE
                                LSDreg_next <= "1001";
                                MSDreg_next <= MSDreg_now - 1;
                        END IF;
                END IF;
        END IF;
END PROCESS;
-------------------------------------------------------------------
-------------------------------------------------------------------
-- Puts updated output onto LSDigit_out and MSDigit_out
-- when the update pin provides a positive transition which is
-- provided every second.  Therefore the output display is a clock
-- counting in seconds.
PROCESS(update, Reset_Flag)
BEGIN
        IF(Reset_Flag = '0')THEN

                Buzer <= '0';
                Done_Flag <= '0';
                LSDreg_now <= LSDigit;
                MSDreg_now <= MSDigit;

        ELSIF(update ='1' AND update'EVENT) THEN

                LSDreg_now <= LSDreg_next;
                MSDreg_now <= MSDreg_next;

                IF(LSDreg_now = "0001" and MSDreg_now = "0000") THEN
                        Buzer <= '1';
                        Done_Flag <= '1';
                END IF;
        END IF;
END PROCESS;
-------------------------------------------------------------------
```

FIGURE 4.1.12 (*Continued*) VHDL code for the counter component.

```
        LSDigit_out <= LSDreg_now;
        MSDigit_out <= MSDreg_now;
END arch;
-- ***************************************************************
-- END OF PROGRAM
```

FIGURE 4.1.12 (*Continued*) VHDL code for the counter component.

```
-- ***************************************************************
-- Program Name        :       sevensegdisplay.vhd
-- File Name           :       desktop\new folder\altera\kitchen timer
-- Programed By        :       Brian Fast
-- Date                :       April 6, 2004
-- Purpose             :       display output in correct form
-- Input    :   Two Four Bit Registers that Represent a unsigned
--              decimal number (0-9)
--              The two regisiters represents the Least Significant Digit
--              and Most SIgnificant Digit
-- Ouput    :   Two seven bit Registers that Represents the decimal
--              number of the the two corresponding binary inputs.
--              the seven bit output will represend the appropriate bit
--              pattern to display the value on a 7 segment LED display
-- Chip     :   10K70RC240-4
-- Board    :   Altera University Program Development Board UP2
-- Software :   Altera Max+Plus v10.22
--
-- NOTE     :   This file will be used as a black box within
--              another file so the I/O pins will not be
--              set within this file.  The I/O signals will
--              be interconnections set within the main program
--              which is where the I/O pins will be set
--              Num             Description                     By              Date
-------------------------------------------------------------------------------------
-- Status :     1.0             Began Initial Program           BRF             4/6/2004
--
--                                          --------------
--                                          |     two      |  input to 7-seg display
--                                          |    seven     |
--              [LSinput]  (4 bits)    ---->|     seg      |---> [LSdisplay]  (7 bits)
--                                          |   Display    |---> [MSdisplay]  (7 bits)
--              [MSinput]  (4 bits)    ---->|              |---> [DPdisplay]  (2 bits)
--                                          |              |
--                                          --------------
--                                    Currently the Program is making the appropriate
--                                    Decimal to 7 seg display conversion.
--
-- ***************************************************************
-- Include Libraries
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE ieee.std_logic_unsigned.ALL;
```

FIGURE 4.1.13 VHDL code for the seven segment display component.

```
-- ************************************************************
-- Declaration of INPUT & OUTPUT variables
-- ENTITY must have a reference which is the same as  the file name

ENTITY sevensegdisplay IS
       PORT(
                      LSinput        :       IN      std_logic_vector(3 DOWNTO 0);
                      MSinput        :       IN      std_logic_vector(3 DOWNTO 0);
                      LSdisplay      :       OUT     std_logic_vector(6 DOWNTO 0);
                      MSdisplay      :       OUT     std_logic_vector(6 DOWNTO 0);
                      DPdisplay      :       OUT     std_logic_vector(1 DOWNTO 0));
END sevensegdisplay;
-- ************************************************************
-- ************************************************************
-- The actual body of the program

ARCHITECTURE arch OF sevensegdisplay IS
       -- Look up table for seven segment LCD
       --                   a
       --                  ---
       --               f | _ | b
       --               e | g | c
       --            dp.    ---
       --                   d
       --
       -- 1 equals off
       -- 0 equals on
       --                                                  gfedcba
       CONSTANT seg7_0: std_logic_vector(6 DOWNTO 0):= "1000000";
       CONSTANT seg7_1: std_logic_vector(6 DOWNTO 0):= "1111001";
       CONSTANT seg7_2: std_logic_vector(6 DOWNTO 0):= "0100100";
       CONSTANT seg7_3: std_logic_vector(6 DOWNTO 0):= "0110000";
       CONSTANT seg7_4: std_logic_vector(6 DOWNTO 0):= "0011001";
       CONSTANT seg7_5: std_logic_vector(6 DOWNTO 0):= "0010010";
       CONSTANT seg7_6: std_logic_vector(6 DOWNTO 0):= "0000010";
       CONSTANT seg7_7: std_logic_vector(6 DOWNTO 0):= "1111000";
       CONSTANT seg7_8: std_logic_vector(6 DOWNTO 0):= "0000000";
       CONSTANT seg7_9: std_logic_vector(6 DOWNTO 0):= "0010000";
       CONSTANT seg7_n: std_logic_vector(6 DOWNTO 0):= "0111111";
       CONSTANT seg7_x: std_logic_vector(6 DOWNTO 0):= "1111111";
       CONSTANT DP_on : std_logic_vector(1 DOWNTO 0):= "00";
       CONSTANT DP_off: std_logic_vector(1 DOWNTO 0):= "11";
BEGIN
       -- reads the value on the Least Significant Input (4 bits)
       -- and then displays the appropriate Binary to Decimal Conversion
       -- the output is then feed to the LCD display.
       -- that conversion is taken care of in the look up table above
       WITH    MSinput SELECT
                      MSdisplay <=   seg7_0          WHEN "0000",
                                     seg7_1          WHEN "0001",
                                     seg7_2          WHEN "0010",
                                     seg7_3          WHEN "0011",
                                     seg7_4          WHEN "0100",
                                     seg7_5          WHEN "0101",
                                     seg7_6          WHEN "0110",
```

FIGURE 4.1.13 (*Continued*) VHDL code for the seven segment display component.

```
                                    seg7_7          WHEN "0111",
                                    seg7_8          WHEN "1000",
                                    seg7_9          WHEN "1001",
                                    seg7_x          WHEN OTHERS;
        -- reads the value on the Most Significant Input (4 bits)
        -- and then displays the appropriate Binary to Decimal Conversion
        -- the output is then feed to the LCD display.
        -- that conversion is taken care of in the look up table above
        WITH    LSinput SELECT
                        LSdisplay <=    seg7_0          WHEN "0000",
                                        seg7_1          WHEN "0001",
                                        seg7_2          WHEN "0010",
                                        seg7_3          WHEN "0011",
                                        seg7_4          WHEN "0100",
                                        seg7_5          WHEN "0101",
                                        seg7_6          WHEN "0110",
                                        seg7_7          WHEN "0111",
                                        seg7_8          WHEN "1000",
                                        seg7_9          WHEN "1001",
                                        seg7_x          WHEN OTHERS;
                        DPdisplay <= DP_off;
END arch;
-- ***************************************************************
```

FIGURE 4.1.13 (*Continued*) VHDL code for the seven segment display component.

CONCLUSION

In conclusion this chapter has gone through the process for designing integrated digital systems using VHDL. Details were provided for the appropriate syntax and for design philosophies, which will help a designer get started. This chapter was intended for someone who has a familiarity with digital and embedded system design.

INFORMATION RESOURCES

Chip supplier: http://www.altera.com/#
Chip supplier: http://www.latticesemi.com/products/fpga/index.cfm?qsmod=1#xpga
Chip supplier: http://www.atmel.com/products/ULC/
Chip supplier: http://www.xilinx.com/
Article about VHDL component programming:
http://www.circuitcellar.com/library/print/0899/Anderson109/8.htm
Software: http://www.electronicstalk.com/news/qui/qui132.html
Overview: http://www.ecs.umass.edu/ece/labs/vlsicad/ece665/spr04/projects/kesava-fpga.pdf
FPAA:
http://bach.ece.jhu.edu/~tim/research/fpaa/fpaa.html
http://www.designw.com/
http://www.ee.ualberta.ca/~vgaudet/fpaa/index.html
http://howard.engr.siu.edu/~haibo/research/mfpa/
http://www.circuitcellar.com/library/newproducts/158/anadigm.html
http://www.ece.arizona.edu/~cmsl/Publications/FPAA/mwc98.pdf
http://www.imtek.uni-freiburg.de/content/pdf/public/2004/becker.pdf

FPAA manufacturer: http://www.anadigm.com/
http://www.zetex.com/
FPGA & FPAA manufacturers list:
http://www.alexa.com/browse/general?catid=57799&mode=general
FPGA & FPAA manufacturer:
http://www.latticesemi.com/products/index.cfm?CFID=4003705&CFTOKEN=67611870

BIBLIOGRAPHY

Brown, S., and Z. Vranesic, "Fundamentals of Digital Logic with Verilog Design," McGraw-Hill, 2003.
Pierzchala, E., "Field-Programmable Analog Arrays," Kluwer Academic Publishers, 1998.
Razavi, B., "Design of Analog CMOS Integrated Circuits," McGraw-Hill, 2001.
Uyemura, J. P., "A First Course In Digital Systems Design," Brooks/Cole Publishing, 1999.