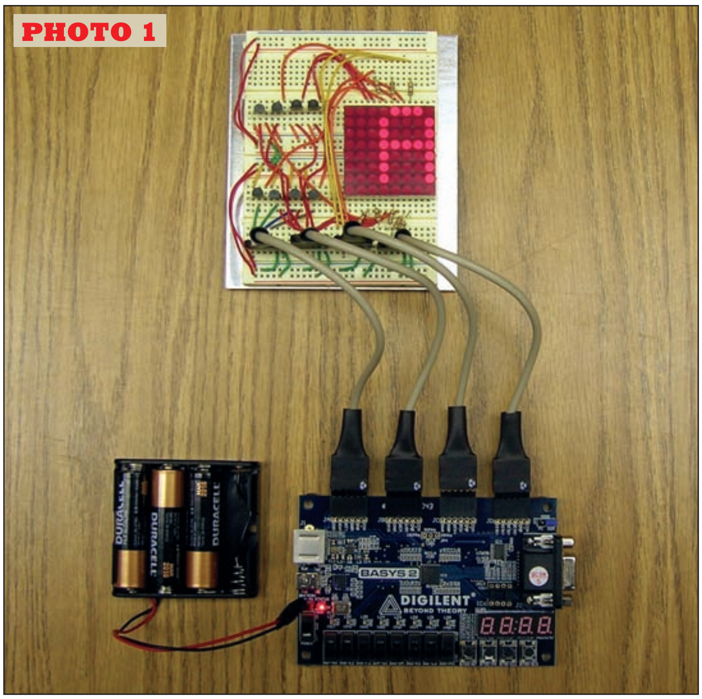
Getting Started With FPGAs — Part 2

by David Ward

In the first article about FPGAs, the reader was introduced to the Digilent Basys2 FPGA trainer and the Xilinx XC3S100E 100K gate FPGA in a 132-pin surface-mount package. The reader was shown how to enter a simple two input AND gate in VHDL, compile the listing, generate a configuration bit file, and download that bit file into the FPGA and test it.

In this second and final article, we will demonstrate a more complex and useful digital design that will use the Basys2 FPGA trainer to control a scrolling message on an 8 x 8 LED matrix display; see **Photo 1**. You will need to use four of the Digilent expansion cables and all of the 16 available FPGA expansion pins to make this circuit operate. The 8 x 8 LED matrix that is used is a Kingbright part number TC15-11SRWA 1.5" dot matrix display from **www.kingbrightusa.com** for $4.76. When you look up the Kingbright matrix, you'll notice that the top surface is not red like in **Photo 1**. If you cover the surface with red tail light repair tape, your letters appear brighter.

PHOTO 1



A complete schematic diagram is shown in **Figure 1**. The cathodes of the LEDs in the matrix are connected to the column pins which are then taken to ground through eight 2N7000 FETs. Notice that no resistors are required when using these FETs in the manner shown here. The gates of the FETs are connected to the eight column outputs from the Basys2 expansion connectors. Only one of these FETs will ever be on at one time because a ring counter will be used to sequentially scan through them one at a time over and over again at a fast rate, giving the illusion that all of the LED columns are lit at the same time. In reality, a maximum of eight LEDs will ever be on at the same time. Each FET will conduct (at the most) 80 mA at one time; eight LEDs x 10 mA each = 80 mA. According to the datasheets they are capable of conducting 200 mA.

The anodes of the LEDs are connected to the row pins on the matrix. The rows will be driven by eight FPGA pins from the Basys2 expansion connectors in series with eight 150 ohm resistors. These eight resistors are necessary to limit the current from each FPGA row pin to 10 mA. The LEDs drop 1.8V when conducting 10 mA. Therefore, 3.3V (a LVTTL logic "1") – 1.8V = 1.5V and 1.5V / 10 mA = 150 ohms of resistance. The FPGA pins can be configured in the UCF file to drive 16 mA. There will not be any problems driving the 64 LEDs in the matrix with the FPGA unless you

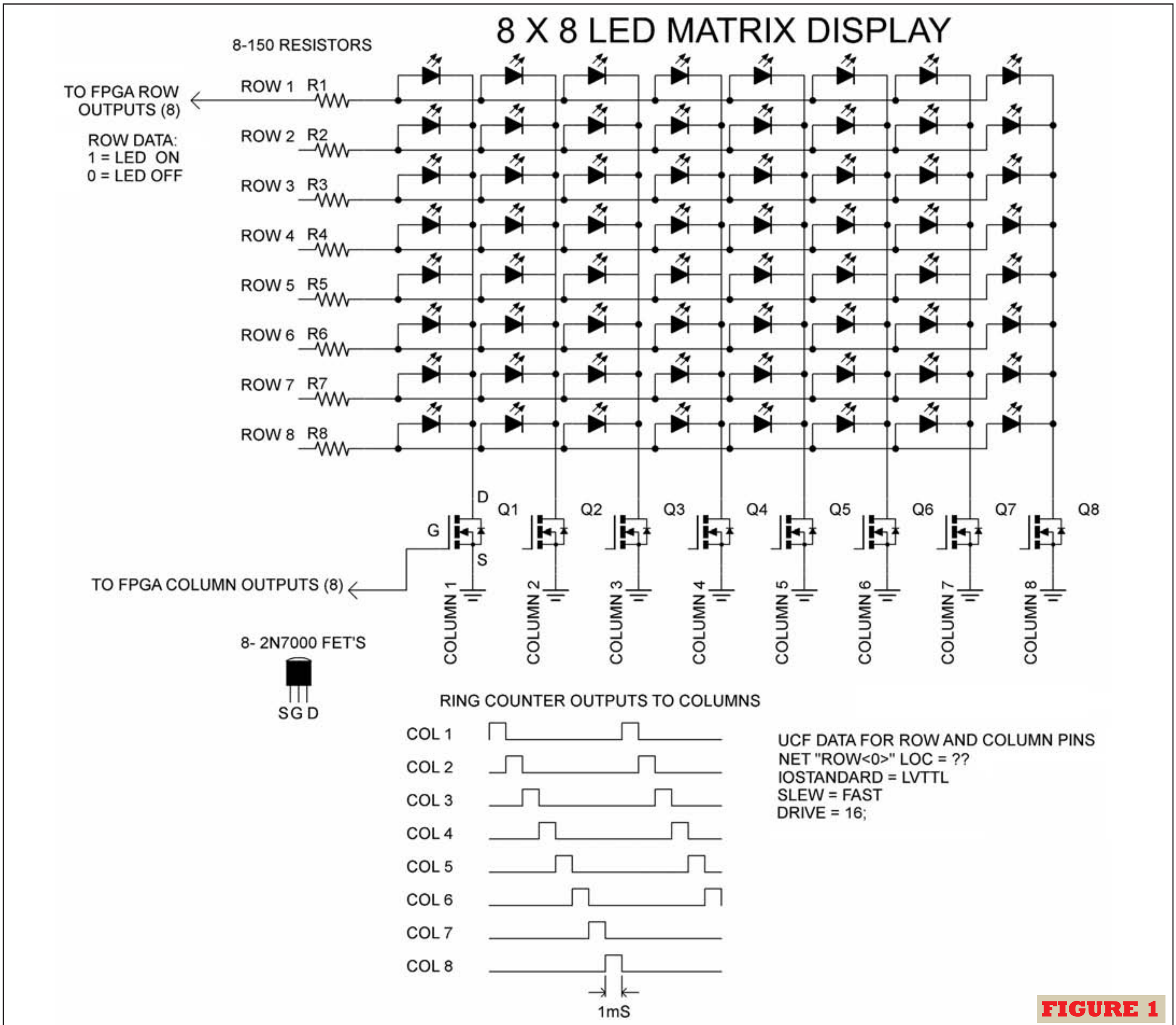


FIGURE 1

attempt to energize more than one column at a time.

Now, let's step through the VHDL and UCF listings line by line and explain what is occurring; see **Listings 1** and **2**. By the way, this code displays the message "A," "B," "C," and a blank as it scrolls from the right side of the matrix over to the left. The code has been commented here and there to help the reader see what is being done. Comments in VHDL are made by typing in two dashes ("—") and then your comments. Comments are not compiled; they are ignored by the Xilinx compiler. So, it won't be necessary to elaborate on the comment lines such as lines 1 through 3.

Lines 5 through 9 are the "Entity" section of the VHDL code listing. This is where ports or actual input and output signals are defined. Line 6 defines an input line named "clk" which is a single bit. The UCF file will direct this to pin B8 of the FPGA where the Basys2 is connected to a 50 MHz clock signal; see line 1 of the UCF file in **Listing 2**. Line 7 defines an eight-bit output port named DISPLAY_C<0> through DISPLAY_C<7>. These are the eight bits that will go out to the columns of the LED matrix. Their locations are defined in lines 3 through 10 of the UCF listing. Line 8 defines an eight-bit output port named DISPLAY_R<0> through DISPLAY_R<7>. These are the eight bits that will go out to the rows of the LED matrix. Their locations are defined on lines 12 through 19 of the UCF listing. Line 9 marks the end of the Entity section.

Lines 12 through line 56 are the "Architecture" section of the VHDL code listing. This is where the logical behavior of the circuit is defined. Lines 14 though 16 set up a signal named ABC that is 256 bits wide. It is initialized with the ones and zeros to display A, B, and C with a final blank screen. Line 17 sets up a signal called low_clk. This signal will be a lower frequency signal derived from the higher 50 MHz signal coming into the clk pin. Line 18 sets up a signal called

```
—Basys2_ABC by David A. Ward May 2011
—Displays ABC, this uses 64 bits per letter data
—and scrolls R to L
—clk info:  50MHZ = 20nS period applied to FPGA
—pin B8

entity Basys2_ABC is
       PORT (clk : IN bit;
       —50MHZ clock coming in to pin B8
       DISPLAY_C : OUT bit_vector (0 TO 7);
       —8 column outputs to ground cathodes
       DISPLAY_R : OUT bit_vector(0 TO 7));
       —8 row outputs to source LED anodes
end Basys2_ABC;

—display info is 64 bits from row1 col1 down, every
—8 bits is one column of data
architecture Behavioral of Basys2_ABC is
—256 bits, 64 bits per full display, 4 displays;
—A, B, C, and a blank
       SIGNAL ABC : bit_vector(255 DOWNTO 0) :=


“00000000011111101001000010010000100100000111111000
000000000000000000000011111110100100101001001010010
010011011000000000000000000000000000111110010000010
100000101000001001000100000000000000000000000000000
000000000000000000000000000000000000000000000000000
00”;
       SIGNAL low_clk : bit;
       —reduced frequency    clock
       SIGNAL ringcounter : bit_vector(7 DOWNTO 0)
       := “00000001”;
       —init ring counter

BEGIN

—clock divider portion to reduce the F of the 50MHZ
—clock down
       PROCESS (clk)
              VARIABLE inc, inc_2 : integer := 0;
              —counter variable initialized to 0
              BEGIN
               IF(clk’EVENT AND clk = ‘1’) THEN
               inc
               := inc + 1; —increment on PGT
               IF(inc = 25000) THEN —toggle the
               low_clk 25,000 X 20nS = 500uS
                                 IF(low_clk = ‘1’) THEN low_clk <=
                                 ‘0’; —toggle reduced clock signal
                                   inc_2 := inc_2 +1; —increment
                                   scroll counter
                                     IF(inc_2 = 125) THEN inc_2
                                     :=
                                     0; —reset counter
                                         ABC <= ABC rol 8;  —time
                                         to scroll, rotate data 8
                                         bits left
                                         ELSE ABC <= ABC;  —not
                                         time to scroll, use past
                                         data
                                     END IF;
                                         ELSE low_clk <= ‘1’;
                                         ringcounter <=
                                         ringcounter rol 1;
                                         —rotate ring counter
                                   END IF;
                              inc := 0;      —clear count
                       END IF;
END IF;

DISPLAY_C <= ringcounter;
—8 lines used to energize 8 FET’s

CASE ringcounter IS
—output letter data depending on switch settings
         WHEN “00000001”  => DISPLAY_R <= ABC(255
                          DOWNTO 248);   —column 1
         WHEN “00000010”  => DISPLAY_R <= ABC(247
                          DOWNTO 240);
         WHEN “00000100”  => DISPLAY_R <= ABC(239
                          DOWNTO 232 );
         WHEN “00001000”     => DISPLAY_R <=
                          ABC(231 DOWNTO 224);
         WHEN “00010000”  => DISPLAY_R <= ABC(223
                          DOWNTO 216);
         WHEN  “00100000” => DISPLAY_R <= ABC(215
                          DOWNTO 208);
         WHEN “01000000”  => DISPLAY_R <= ABC(207
                         DOWNTO 200);
         WHEN “10000000” => DISPLAY_R <= ABC(199
                         DOWNTO 192);  —column 8
       WHEN OTHERS => DISPLAY_R <= “00000000”;
     END CASE;
   END PROCESS;
end Behavioral;
```

LISTING 1

ringcounter which is eight bits wide and is initialized to “00000001.” This signal will be constantly rotated left by one bit to drive the column scanning. Line 20 marks the beginning of the Architecture section after the signals have been defined.

Line 23 is the label for a process named “clk.” A process operates sequentially in the FPGA instead of in a parallel manner. Line 24 defines two integer variables to be used in the process: inc and inc_2. They are both initialized to a value of zero. The first one, inc, will be incremented on every positive going transition of the 50 MHz clock. The second one, inc_2, will be incremented off of the low_clk signal to control the scrolling of the letters across the matrix. Line 25 marks the beginning of the clk process.

Lines 26 through 39 are four nested IF, THEN, ELSE statements. Line 26 will increment inc by one on every positive going transition (PGT) of the 50 MHz clock; this will occur every 20 nS. Lines 27 and 28 determine that if inc has gotten up to 25,000 or that 500 µS have elapsed, then toggle the low_clk signal. That is, if low_clk was previously a 1, then make it a 0. If it was a 0, then make it a 1. Line 29 increments inc_2 by one every time low_clk is toggled.

Line 30 determines that if inc_2 has gotten up to 125, then reset inc_2 back to zero. Then, on line 31, rotate the 256-bit signal ABC left by eight places. Line 32 determines if inc_2 has not reached 125 yet, then don’t rotate ABC left eight places; leave it as it was.

Line 33 is the END IF for line 30. Line 34 is the Else for line 28 for toggling the low_clk signal. Line 35 rotates the ringcounter left by one place; this is for scanning the columns. Line 36 is the END IF for line 28. Line 37 resets the inc counter back to zero. Line 38 is the END IF for line 27. Line 39 is the END IF for line 26.

Line 41 is where the signal ringcounter is actually sent out to the eight pins of DISPLAY_C<0> through DISPLAY_C<7> which is where the matrix column scanning takes place.

Lines 43 through 53 are a CASE structure. The CASE structure will only find one of the lines from line 44 through 51 true at a time. Since the CASE structure is comparing to ringcounter to determine which line to execute, it will step sequentially through them from line 44 then on to line 45, etc., until reaching line 51. Since ringcounter is rotated every 1 mS, then the CASE structure will step through its eight choices every 8 mS. When the CASE structure finds a match — such as in line 44 when ringcounter = "00000001" — then it will send out the upper eight bits of ABC to DISPLAY_R<0> through DISPLAY_R<7>. This is where the eight LEDs on the eight rows are energized at the anodes and the FET from column 1 is turned on to ground all of the cathodes from column 1. So, the CASE structure is where the data actually gets out to the matrix. As the CASE structure cycles through the eight ringcounter values, it will display an entire 8 x 8 image.

This happens so rapidly that the human eye cannot see it is actually scanning across the matrix and only lighting a maximum of eight LEDs at any one time. If the 256-bit signal ABC is rotated left by eight bits or one column every so often, then you can see the letters slowly scroll across the matrix from right to left. Lines 54 and 56 mark the end of the process and the Architecture sections.

If you look at the UCF listing, you'll see that it does several other things than just locate or "LOC" signals from the VHDL listing with the physical FPGA pins. The UCF also sets the IOSTANDARD to LVTTL. There are over 20 I/O standards available in this FPGA. The LVTTL stands for low voltage TTL, which means that a logic "1" is not 5.0V but 3.3V. The slew rate or the speed at which the signals transition from a 1 to a 0 and vice versa are set to FAST. They can also be set to SLOW if the receiving devices require it. Finally, the DRIVE is set to 16 mA. The drive can be set as low as 2 mA and a maximum of 16 mA.

You will probably want to make up your own messages since watching "ABC" slowly scrolling across isn't very exciting. By the way, entering all 256 of your 1s and 0s for your message isn't too convenient, but it makes the VHDL listing shorter and easier to follow. **Figure 2** shows you how the data is generated for the letters so you can develop your own messages. A blank form is included in the article downloads.

An easier way to generate your messages is to develop them column by column in Notepad and then cut and paste them into your code; see **Listing 3**. If you want a longer message, the only thing that needs to be changed is to paste your extra letters into the end of the data at line 16 and then increase the number (XXX) in line 14 by 64 for each additional character, SIGNAL ABC: BIT_VECTOR (XXX DOWNTO 0). Of course, if you don't change the numbers in the CASE section (lines 43 through 53), the message will start somewhere in the middle during the first pass through, but it will appear correct after one pass.

Hopefully, you've enjoyed these articles on FPGAs and it has given you a starting point to learn more about them and VHDL. Again, a good tutorial for VHDL is "The Low-Carb VHDL Tutorial" by Bryan Mealy available on the Internet. **SV**

LISTING 2

```
NET "clk" LOC = B8;

NET "DISPLAY_C<0>" LOC = B7 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_C<1>" LOC = C5 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_C<2>" LOC = B6 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_C<3>" LOC = C6 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_C<4>" LOC = B5 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_C<5>" LOC = J3 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_C<6>" LOC = A3 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_C<7>" LOC = B2 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;

NET "DISPLAY_R<0>" LOC = A9 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_R<1>" LOC = A10 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_R<2>" LOC = D12 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_R<3>" LOC = B9 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_R<4>" LOC = C9 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_R<5>" LOC = C13 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_R<6>" LOC = C12 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
NET "DISPLAY_R<7>" LOC = A13 | IOSTANDARD = LVTTL | SLEW = FAST | DRIVE = 16;
```

LISTING 3

```
A
00000000
01111110
10010000
10010000
10010000
01111110
00000000
00000000

B
00000000
11111110
10010010
10010010
10010010
01111100
00000000
00000000

C
00000000
01111100
10000010
10000010
10000010
01000100
00000000
00000000
```

FIGURE 2

8 X8 LED MATRIX FOR LETTER "A"

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8		D7	D6	D5	D4	D3	D2	D1	D0
Row 1				*					Row 1 data	0	0	0	0	0	0	0	0
Row 2			*		*				Row 2 data	0	0	1	1	1	1	1	0
Row 3		*				*			Row 3 data	0	1	0	1	0	0	0	0
Row 4		*	*	*	*	*			Row 4 data	1	0	0	1	0	0	0	0
Row 5		*				*			Row 5 data	0	1	0	1	0	0	0	0
Row 6		*				*			Row 6 data	0	0	1	1	1	1	1	0
Row 7		*				*			Row 7 data	0	0	0	0	0	0	0	0
Row 8									Row 8 data	0	0	0	0	0	0	0	0

"B"

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8		D7	D6	D5	D4	D3	D2	D1	D0
Row 1		*	*	*	*				Row 1 data	0	0	0	0	0	0	0	0
Row 2		*				*			Row 2 data	1	1	1	1	1	1	1	0
Row 3		*				*			Row 3 data	1	0	0	1	0	0	1	0
Row 4		*	*	*	*	*			Row 4 data	1	0	0	1	0	0	1	0
Row 5		*				*			Row 5 data	1	0	0	1	0	0	1	0
Row 6		*				*			Row 6 data	0	1	1	1	1	1	0	0
Row 7		*	*	*	*				Row 7 data	0	0	0	0	0	0	0	0
Row 8									Row 8 data	0	0	0	0	0	0	0	0

"C"

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8		D7	D6	D5	D4	D3	D2	D1	D0
Row 1			*	*	*				Row 1 data	0	0	0	0	0	0	0	0
Row 2		*				*			Row 2 data	0	1	1	1	1	1	0	0
Row 3		*							Row 3 data	1	0	0	0	0	0	1	0
Row 4		*							Row 4 data	1	0	0	0	0	0	1	0
Row 5		*							Row 5 data	1	0	0	0	0	0	1	0
Row 6		*				*			Row 6 data	0	1	0	0	0	1	0	0
Row 7			*	*	*				Row 7 data	0	0	0	0	0	0	0	0
Row 8									Row 8 data	0	0	0	0	0	0	0	0