





© 2001 by X Engineering Software Systems Corp., Apex, North Carolina 27502

All rights reserved. No part of this text may be reproduced, in any form or by any means, without permission in writing from the publisher.

The author and publisher of this text have used their best efforts in preparing this text. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this text. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

XESS, XS40, and XS95 are trademarks of X Engineering Software Systems Corp. XILINX, Foundation, XC4000, and XC9500 are trademarks of XILINX Corporation. Other product and company names mentioned are trademarks or trade names of their respective companies.

The software described in this text is furnished under a license agreement. The software may be used or copied under terms of the license agreement.



# **Using RAM**

## In this chapter you will learn how to:

- Interface to an external RAM with a programmable device.
- Use the internal RAMs found in the XC4000 FPGAs.
- Create RAM modules using the Core Generator.

## RAM

Eventually you will need to incorporate RAM into one of your designs. While you can build multi-bit registers from the flip-flops in a CPLD or FPGA, it is more efficient to use an external RAM chip or a specialized internal RAM-block when you need to store larger amounts of data.

In this chapter we will build a simple design that reads a set of data bytes from RAM, writes the 2'scomplement of the byte values back into the RAM, sums the complemented data values and then displays the sum on the seven-segment LED. We will do two different versions of this design:

- 1. The first version will store the data values in the external asynchronous, byte-wide RAM found on the XS40 and XS95 Boards.
- 2. The second version will store the data values using the internal synchronous, distributed RAM contained in the XC4000 FPGA on the XS40 Board.

## Using an External Asynchronous RAM

The first version of the RAM summation circuit has the design hierarchy shown in Figure 12. The root module of the design manages the interface to the external asynchronous RAM and sums the data while the lower-level module displays a four-bit hexadecimal value on a seven-segment display.



Figure 12: Design hierarchy for a logic circuit that displays the summation of data in the RAM.

Each of these modules is described by a VHDL file stored in the *dsgn5\_1* project directory that was created as follows.

New Proje	ect		×
Name:	dsgn5_1		ОК
Directory:	C:\PRAG21I		Cancel <u>B</u> rowse
Туре:	F2.1i	•	<u>H</u> elp
Flow:	C <u>S</u> chematic	● HDL	

After the VHDL files for the modules were created and added to the project, the **Project Navigator** window appears as follows. Now I will describe the contents of each VHDL file.

🐌 dsgn5_1 - 95108-20PC84 - Pro	oject Manager 📃 🗖	×			
<u>File D</u> ocument <u>V</u> iew <u>Project</u> <u>Synthesis</u> <u>Implementation</u> <u>T</u> ools <u>H</u> elp					
D 🖻 🖯 🐽 🔿 🕤	🖻 🗭 🖻 🖶 💦				
Files Versions	Flow Contents Reports				
☐ ☐ dsgn5_1 + ⓓ / leddcd.vhd ① 個 / ramsum.vhd ☐ dsgn5_1	dsgn5_1				
	📰 😵 🍋				
	DESIGN ENTRY				
	SYNTHESIS / SIMULATION				
		<b>•</b>			
Dpm : Done					
Pcm : Update: c:\prag21i\dsgn5_1\ramsum.vhd (0, 0)					
Pcm : Synopsys server initialization					
Console / HDL Errors / HDL Warnings / HDL Messages /					
Ready					

#### The LED Decoder Module

This LED decoder circuit is almost identical to the one in Chapter 3 except for the addition of a blanking input signal that causes all the LED segments to turn off. This signal will be used to blank the display to separate the digits when displaying a multi-digit hexadecimal number.

The VHDL code for the LED decoder is shown in Listing 4. This code is stored in the leddcd.vhd file in the *dsgn5\_1* project directory.

```
Listing 4: VHDL code for the seven-segment LED decoder module.
```

```
1
    library IEEE;
2
    use IEEE.std logic 1164.all;
3
    use IEEE.numeric std.all;
4
5
    package leddcd pckg is
6
7
    component leddcd
8
      port (
9
        blank: in STD LOGIC; -- active-high blanking input
10
             : in UNSIGNED (3 downto 0);
        d
```

```
11
             : out STD LOGIC VECTOR (6 downto 0)
        S
12
      );
13
    end component;
14
15
    end leddcd pckq;
16
17
18
    library IEEE;
19
    use IEEE.std logic 1164.all;
20
    use IEEE.numeric std.all;
21
22
    entity leddcd is
23
      port (
24
        blank: in STD LOGIC; -- active-high blanking input
25
        d : in UNSIGNED (3 downto 0);
26
             : out STD LOGIC VECTOR (6 downto 0)
        S
27
      );
28
    end leddcd;
29
30
    architecture leddcd arch of leddcd is
31
    signal s tmp: STD LOGIC VECTOR(6 downto 0);
32
    begin
33
      with d select
34
      s tmp <= "1110111" when "0000", -- 0
35
                "0010010" when "0001", -- 1
36
                "1011101" when "0010", -- 2
37
                "1011011" when "0011", -- 3
38
                "0111010" when "0100", -- 4
39
                "1101011" when "0101", -- 5
40
                "1101111" when "0110", -- 6
41
                "1010010" when "0111", -- 7
42
                "11111111" when "1000", -- 8
43
                "1111011" when "1001", -- 9
44
                "1111110" when "1010", -- A
45
                "0101111" when "1011", -- b
46
                "1100101" when "1100", -- C
47
                "0011111" when "1101", -- d
48
                "1101101" when "1110", -- E
49
                "1101100" when others; -- F
50
51
      -- zero the outputs if the blanking signal is high,
52
      -- otherwise output the LED digit bit pattern
53
      s \leq "0000000" when blank='1' else s tmp;
54
    end leddcd arch;
```

#### The Root Module

The root module sequences through three main phases:

**Phase 1:** Starting from an upper address of RAM and proceeding to address zero, the value stored at each RAM address is read and the two's-complement is computed and written back to the same address.

- **Phase 2:** Restarting from the upper address and proceeding to address zero, each value is read from RAM and added to a sum register.
- **Phase 3:** The sum is displayed on the seven-segment LED by blanking the LED segments for a long interval to signal the start of the sum, then the hexadecimal digit for the upper four bits of the sum are displayed, then the LEDs are blanked for a shorter interval and then the hexadecimal digit for the lower four bits is displayed. Then this four-step display process repeats.

The VHDL code for the root module is in the ramsum.vhd (Listing 5). Some highlights from the code are given below.

- Line 4: The root module accesses the component declaration for the LED decoder by using the leddcd\_pckg package that is part of the WORK library. The WORK library is an *implicit library* that has every project module as a member. We could have explicitly created a library and added the leddcd.vhd file to it as we did in Chapter 3, but using the WORK library is a bit simpler.
- Lines 6–17: The interface to the design is declared here. The reset and clock inputs drive the actions of the state machine that controls the operation of the circuit. Data is passed to and from the RAM using the address and data buses along with the chip-enable, write-enable and output-enable control signals. (Note that the RAM data bus is declared as an inout since the same signals are used to get data from the RAM as to send data to it.)
- Lines 20–21: The four-bit RAM address register is declared on these lines as well as the constant for the address of the upper end of the RAM data that will be summed. For this example, the circuit will complement and sum eleven bytes of data from address zero to ten, inclusive.
- Lines 22–23: Two registers that are the same width as the RAM data bus are declared here. One register holds the current value read from the RAM while the other holds the sum of the RAM data values.
- Lines 24–27: These lines declare a time delay register and the constants for the time intervals involved with the display of the hexadecimal digits in the sum.
- Lines 28–29: These lines declare a four-bit bus for sending the hexadecimal digit to the LED decoder and a control signal to force the LED display to blank.
- Lines 31–33: The nine states of the state machine are declared along with a register that holds the current state.
- Line 37: This is the start of the process that computes the next state for the state machine given the current state, RAM address and delay timer value. The values for the RAM address, data and control signals are also generated in this process.
- Lines 40–50: The default outputs for this process are defined here. The state, RAM address, summation and RAM byte registers all retain their current values unless explicitly changed within the process body. The delay register is decremented. The LED display is blanked. The RAM is enabled, but any

read or write operations are disabled. The data bus is tristated to remove any chance of contention between the FPGA or CPLD and the RAM.

- Line 52: This is the start of the case statement that computes the outputs from this process based on the current state stored in the st r register.
- Lines 53–55: The init state initializes the state machine for the start of the loop that complements the contents of RAM. The address register is set to point to the upper bound of the RAM data range and the state machine is moved to the start of the two's-complement loop (invertr).
- Lines 56–59: The invertr state activates the outputs of the external RAM. The FPGA or CPLD will tristate its own outputs to the RAM data bus so that there is no contention (the default statement handles this). The value from RAM is complemented and stored in the RAM byte register. Then the state machine is moved to the state where the complemented data is written back to the RAM (invertw).
- Lines 60–63: The invertw state activates the write-enable of the external RAM during the second half of the clock cycle (when the clock is low). The value in the RAM byte register is sent out to the RAM on the data bus. Then the state machine is moved to a NOP state to terminate the RAM write (invertnop).
- Lines 64–74: The invertnop state keeps the RAM address stable while the writeenable returns to its quiescent state (the default operation statements handle this). The value in the RAM byte register remains on the output bus to the RAM for the same reason. If the current RAM address is zero indicating the complementation loop is finished, then the RAM address is reloaded with the starting address of the (now complemented) RAM data. Then the state machine is moved to the start of the summation loop (add). If not all the RAM data has been complemented yet, then the RAM address is decremented and the state machine returns to the start of the complementation loop (invertr).
- Lines 75–85: The add state activates the outputs of the external RAM. The value from RAM is added to the summation register. If the current RAM address is zero indicating the summation loop is finished, then the time delay register is loaded with the initial blanking interval for the LED display. Then the state machine is moved to the start of the display loop (display\_blank). If all the RAM data has not been summed, then the RAM address is decremented and the state machine stays in the add state.
- Lines 86–91: The display\_blank state blanks the LED display and decrements the delay timer (the default operation statements handle this). Once the delay timer reaches zero, it is reloaded with the time interval for display of a digit and the state machine is moved into the display upper digit state.
- Lines 92–99: The display\_upper\_digit state unblanks the LED display and sends the upper four bits of the sum to the LED decoder. The delay timer is also decremented. Once the delay timer reaches zero, it is reloaded with the time interval for blanking the display between digits and then the state machine is moved into the display interdigit state.

- Lines 100–105: The display\_interdigit state blanks the LED display and decrements the delay timer (the default operation statements handle this). Once the delay timer reaches zero, it is reloaded with the time interval for display of a digit and the state machine is moved into the display\_lower\_digit state.
- Lines 106–113: The display\_lower\_digit state unblanks the LED display and sends the lower four bits of the sum to the LED decoder. The delay timer is also decremented. Once the delay timer reaches zero, it is reloaded with the time interval for blanking the display before the sum is displayed and then the state machine is moved into the display\_blank state.
- Lines 120–134: This process updates the state, address, data, sum and time delay registers with their new values on the rising edge of the clock. The reset input synchronously clears the sum register and transfers the state machine into its starting state.
- Line 136: The four-bit RAM address is padded with leading zeroes to form the complete address passed to the external RAM.
- Line 137: The LED decoder is passed the blanking signal and the four-bit hexadecimal code for the digit to be displayed. The outputs of the LED decoder drive the seven-segment LEDs.

```
Listing 5: VHDL code for the root module.
```

```
1
     library IEEE;
 2
     use IEEE.std logic 1164.all;
 3
     use IEEE.numeric std.all;
 4
     use WORK.leddcd pckg.all;
 5
 6
    entity ramsum is
 7
      port (
 8
        rst : in STD LOGIC;
                                                    -- reset
 9
                                                    -- clock
          clk : in STD LOGIC;
                : out UNSIGNED(16 downto 0);
10
                                                   -- RAM address bus
          а
                : inout UNSIGNED(7 downto 0);
11
          d
                                                     -- RAM data bus
         ce_n : out STD_LOGIC;-- RAM chip-enablewe_n : out STD_LOGIC;-- RAM write-enableoe_n : out STD_LOGIC;-- RAM output-enable
12
13
14
15
          s : out STD LOGIC VECTOR(6 downto 0) -- outputs to LED segments
16
        );
17
     end ramsum;
18
19
     architecture ramsum arch of ramsum is
20
     signal addr r, next addr: UNSIGNED(3 downto 0); -- address register
21
     constant maxaddr : UNSIGNED := TO UNSIGNED(10,addr r'length); -- upper address
22
     signal b r, next b : UNSIGNED(d'length-1 downto 0); -- stores byte from RAM
23
     signal sum r, next sum : UNSIGNED(d'length-1 downto 0); -- stores sum of RAM bytes
24
     signal delay r, next delay : UNSIGNED(22 downto 0); -- delay counter
25
     constant blank dly : UNSIGNED := TO UNSIGNED(5 000 000,delay r'length);
26
     constant interdigit dly : UNSIGNED := TO UNSIGNED(1 600 000,delay r'length);
27
     constant digit dly : UNSIGNED := TO UNSIGNED(2 500 000,delay r'length);
28
     signal digit : UNSIGNED(3 downto 0); -- LED hex digit to display
signal blank : STD_LOGIC; -- LED digit blanking signal
29
30
     -- states for the state machine
31
     type state is (init, invertr, invertw, invertnop, add, display blank,
                                                                                       344
     © 2001 by XESS Corp.
```

```
32
             display upper digit, display interdigit, display lower digit);
33
     signal st_r, next_st : state; -- state register
34
     begin
35
36
     -- this process computes the actions of the state machine in each state
37
     process(clk,st r,addr r,sum r,b r,delay r,d)
38
     begin
39
        -- default outputs unless otherwise specified
40
        next st <= st r;-- next state is the same as the current state
41
        next addr <= addr r; -- don't change the RAM address</pre>
42
        next sum <= sum r; -- don't update the sum register</pre>
43
        next b <= b r; -- don't reload the RAM byte register
44
        next delay <= delay r-1; -- decrement the delay counter
45
        digit <= TO UNSIGNED(0,digit'length); -- output a '0' LED digit</pre>
46
        blank <= '1'; -- blank the LED display
47
        ce n <= '0'; -- enable the RAM
48
        we n <= '1'; -- don't write to the RAM
49
        oe n <= '1'; -- don't read from the RAM
50
              <= (others=>'Z'); -- tristate the but to the RAM
        d
51
52
        case st r is -- case statement for the state machine
53
        when init => -- initialization state
54
         next_addr <= maxaddr; -- start inverting from the upper address</pre>
55
          next st <= invertr; -- enter the RAM inversion loop</pre>
56
        when invertr => -- read the contents of the RAM location
57
          oe n <= '0'; -- enable the RAM outputs
58
           next b <= TO_UNSIGNED(0,next_b'length) - d; -- invert byte from RAM</pre>
59
           next st <= invertw; -- go to RAM-write state</pre>
60
        when invertw => -- write inverted byte value into same RAM location
61
          we_n <= clk; -- write RAM in 2nd half of clock cycle
d <= b_r; -- output inverted byte value to RAM</pre>
62
63
           next st <= invertnop; -- go to RAM no-op state</pre>
64
        when invertnop => -- terminate RAM-write cleanly
65
           d <= b r; -- maintain output of inverted byte value to RAM
66
           if addr r = TO UNSIGNED(0, addr r'length) then
67
             -- reached the lower address of the RAM data
68
             next addr <= maxaddr; -- reload register with upper address</pre>
69
             next st <= add; -- enter the summation loop</pre>
70
           else
71
             -- haven't inverted all the RAM data yet
72
             next addr <= addr r - 1;-- address the next RAM location</pre>
73
             next st <= invertr; -- return to beginning of the inversion loop
74
           end if;
75
                       -- sum the inverted data from RAM
        when add =>
76
           oe_n <= '0'; -- enable the RAM outputs</pre>
77
           next_sum <= sum_r + d; -- add the RAM data to the sum</pre>
78
           if addr r = TO UNSIGNED(0, addr r'length) then
79
            -- reached the lower address of the RAM data
80
             next delay <= blank dly;-- load display interval counter</pre>
81
             next_st <= display_blank; -- now display the sum</pre>
82
           else
83
             -- haven't summed all the RAM data yet so stay in this state
84
             next_addr <= addr_r - 1;-- address the next RAM location</pre>
85
           end if;
86
        when display blank => -- blank the display
87
           if delay r = TO UNSIGNED(0, delay r'length) then
88
             -- initial display blanking is complete
89
             next delay <= digit dly; -- load digit display interval</pre>
90
             next st <= display upper digit; -- now display the upper sum digit
```

```
91
           end if;
92
         when display upper digit => -- display the upper digit of the sum
93
           blank <= '0';</pre>
                                        -- activate the LED
94
           digit <= sum r(7 downto 4); -- display the upper 4-bits of the sum
95
           if delay r = TO UNSIGNED(0, delay r'length) then
96
              -- upper digit display is complete
97
              next delay <= interdigit dly; -- load inter-digit blanking interval
98
              next st <= display interdigit; -- blank the display between digits
99
           end if;
100
         when display interdigit => -- blank the display between sum digits
101
           if delay r = TO UNSIGNED(0, delay r'length) then
102
              -- inter-digit display blanking is complete
103
              next delay <= digit dly; -- load digit display interval</pre>
104
              next st <= display lower digit; -- now display the lower sum digit
105
           end if;
106
         when display lower digit => -- display the lower digit of the sum
107
           blank <= '0';
                                        -- activate the LED
108
           digit <= sum_r(3 downto 0); -- display the lower 4-bits of the sum</pre>
109
           if delay r = TO UNSIGNED(0, delay r'length) then
110
              -- lower digit display is complete
111
              next delay <= blank dly; -- load blanking interval between loops
112
              next st <= display blank; -- loop and display the sum again
113
           end if;
114
         when others = -- error state
115
           next st <= init; -- re-initialize the state machine on an erroneous state
116
         end case;
117
      end process;
118
119
      -- this process updates the registers on every rising clock edge
120
      process(clk)
121
     begin
122
        if clk'event and clk='1' then -- trigger on rising clock edge
123
           if rst='1' then -- synchronous reset
124
              st r <= init;</pre>
125
             sum r <= TO UNSIGNED(0, sum r'length);</pre>
126
           else -- update the registers
127
             st r <= next st;
128
             sum r <= next sum;
129
             addr_r <= next_addr;
130
             b r <= next b;
131
             delay r <= next delay;</pre>
132
           end if;
133
         end if;
134
      end process;
135
136
      a <= "00000000000" & addr r;
137
      u1: leddcd port map(blank=>blank, d=>digit, s=>s);
138
139
      end ramsum arch;
```

Figure 13 shows the waveforms for the phase when the RAM data is complemented. RAM address N is output at the start of the invertr clock cycle and the RAM outputenable is activated. The data stored at address N is output by the RAM to the FPGA or CPLD where it is complemented and stored into the RAM byte register at the start of the invertw cycle. The complemented value in the byte register is sent back to the RAM and the RAM write-enable is pulsed low during the second half of the invertw cycle, thus writing the complemented data back to address N. The RAM address and data are



held stable during the following invertnop cycle and then the entire operation is repeated for RAM address N-1.

#### Figure 13: Timing waveforms for the asynchronous RAM summation circuit.

The timing waveforms illustrate the fundamental principles involved when writing to an asynchronous RAM:

- 1. The address to the RAM must be held stable during the entire time the writeenable is active-low. Otherwise, data may be erroneously written to some other address instead of, or in addition to, the desired address. That's because write operations to an asynchronous RAM occur as long as the write-enable is low, not just on an edge transition.
- 2. The data to the RAM must be held stable at the rising edge of the write-enable pulse. Otherwise, an incorrect data value may be written to the RAM address.

For our design, note that the RAM address is stable for an entire cycle both before and after the write-enable pulse, and the RAM data is stable for an entire cycle before and after the rising edge of the write-enable pulse. This allows a large setup time for the address and data before the write-enable pulse, and provides a large hold time after the pulse.

### Synthesizing and Implementing the Design

Once the modules are checked for syntax and any errors are removed, we can run the synthesis and implementation tools to create the configuration bitstream for the FPGA or CPLD. Click on the Implementation icon to run the synthesizer and the implementation tools sequentially.

🐌 dsgn5_1 - 95108-20PC84 - Pro	iject Manager	. 🗆 🗵			
<u>File D</u> ocument <u>V</u> iew <u>Project</u> <u>Synthesis</u> <u>Implementation</u> <u>T</u> ools <u>H</u> elp					
<u> </u>					
Files Versions	Flow Contents Reports				
□· □ dsgn5_1 +· ⓐ/ leddcd.vhd ⊡· ⓐ/ ramsum.vhd	dsgn5_1				
- 🖯 dsgn5_1	DESIGN ENTRY				
	· · · · · · · · · · · · · · · · · · ·				
	SYNTHESIS ?				
		-			
Dpm : Done	1)romoum uhd (0, 0)	<b>_</b>			
Pcm : Update: c:\prag21i\dsgn5 Pcm : Synopsys server initializa		Ŧ			
Console / HDL Errors / HDL Warnings / HDL Messages /					
Ready					

We will target this design to the XS95 Board, so set the target device to be an XC95108PC84 with a -20 speed grade. Then select the **ramsum** module as the top-level module for the design.

Synthesis/Implementation se	ettings	×				
Top level: ramsum Version name: leddcd ramsum Synthesis Settings:	▼ SET	<u>R</u> un OK <u>C</u> ancel				
- Target Device		<u>H</u> elp				
Family: XC9500	-					
Device: 95108PC84	Speed:	-20 💌				
Edit Synthesis/Implementation constraints						
View Estimated Performance	after Optimiza	tion				
Auto Run Implementation to Physical Implementation settin Revision name: Control Files:		<u>Options</u>				

Next, we need to set the options for the implementation tools.

Synthesis/Imp	lementatio	n settin	gs	×
Top level:	ramsum		-	<u>R</u> un
Version name:	ver1			OK
Synthesis Settin	igs:		SET	<u>C</u> ancel
				<u>H</u> elp
– Target Device	э			
Family: XC9	500	•		
Device: 951	08PC84	-	Speed:	-20 💌
Edit Synthes	is/Implemen	tation co	nstraints	
🔲 View Estimat	ted Performa	nce afte	r Optimiz	ation
🗹 Auto Run I	mplementati	on tools		
Physical Imple	ementation s	ettings —		
Revis	ion name:	rev1		<u>Options</u>
Contr	ol Files:	SE	T	43

Click on the Area button to make the CPLD fitting tool emphasize logic efficiency over operational speed. This option setting is necessary because the design uses a large time delay counter and several comparators that make it difficult to fit into an XC95108 CPLD. Click on the OK button to close the window.

Op	tions				×
	- Implementation Options-				
	CPLD Optimization Style	e:			
	Area	C <u>B</u> alanced	O <u>S</u> peed	C <u>U</u> ser Defined	
		Optimize Density		✓ Edit Options	
	Simulation Options				
	Si <u>m</u> ulation:	Foundation EDIF		E <u>d</u> it Options	
			OK	Cancel <u>H</u> elp	

Next, click on the SET button so we can specify the constraint file that lists the pin assignments for the XS95 Board.

Synthesis/Implementation settings	×				
Top level: ramsum	<u>R</u> un				
Version name: ver1	ОК				
Synthesis Settings: SET	<u>C</u> ancel				
	<u>H</u> elp				
Target Device					
Family: XC9500					
Device: 95108PC84 Speed:	-20 💌				
Edit Synthesis/Implementation constraints					
View Estimated Performance after Optimization					
Auto Run Implementation tools					
Physical Implementation settings					
Revision name: rev1	<u>Options</u>				
Control Files: SET					

Settings D	×
Synthesis Settings Implementation control files	1
Current Revision Control File Settings:	
Use Constraints file from: None	
Copy Guide file from:	
Copy Floorplan files from: None	
Current Revision Control file use:	
Enable Guided MAP and PAR	
Enable Floorplanning	
OK Cancel <u>H</u> elp	

Select the Custom entry in the drop-down list of constraint files.

The **Custom** window should appear with the dsgn5\_1.ucf file already in the Constraints File field. If not, click on the Browse button, find this file in the top-level directory of the **dsgn5\_1** project and select it. Then click on the OK button.

Custom	×
Constraints File: dsgn5_1.ucf	Browse
<u> </u>	Help

The dsgn5\_1.ucf file should specify the assignments for the FPGA or CPLD pins that connect to the clock, reset, seven-segment LED and RAM address, data and control pins as shown in Figure 14. The pin assignments for the XS95 Board (which is our target for this example) are shown in Listing 6. The equivalent pin assignments for the XS40 Board are given in Listing 7.



Figure 14: Connection of the external RAM, programmable oscillator, parallel port, and LED digit to the pins of the FPGA or CPLD on the XS40 or XS95 Board.

#### Listing 6: Pin assignments for the XS95 Board.

<pre># pin assignemnts for the XS95 Board</pre>	
<pre>net clk loc=p9; # clock from programmable osc.</pre>	
<pre>net rst loc=p46; # reset from data pin D0 of parallel port</pre>	
net d<0> loc=p44; # RAM data pin D0	
net d<1> loc=p43; # RAM data pin D1	
net d<2> loc=p41; # RAM data pin D2	
net d<3> loc=p40; # RAM data pin D3	
net d<4> loc=p39; # RAM data pin D4	
net d<5> loc=p37; # RAM data pin D5	
net d<6> loc=p36; # RAM data pin D6	
net d<7> loc=p35; # RAM data pin D7	
net a<0> loc=p75; # RAM address pin A0	
net a<1> loc=p79; # RAM address pin A1	
net a<2> loc=p82; # RAM address pin A2	
net a<3> loc=p84; # RAM address pin A3	
net a<4> loc=p1;  # RAM address pin A4	
net a<5> loc=p3; # RAM address pin A5	
net a<6> loc=p83; # RAM address pin A6	
net a<7> loc=p2; # RAM address pin A7	
net a<8> loc=p58; # RAM address pin A8	
net a<9> loc=p56; # RAM address pin A9	
net a<10> loc=p54;  # RAM address pin A10	
net a<11> loc=p55;  # RAM address pin A11	
net a<12> loc=p53;  # RAM address pin A12	
net a<13> loc=p57; # RAM address pin A13	
net a<14> loc=p61;  # RAM address pin A14	
net a<15> loc=p34;  # RAM address pin A15	
net a<16> loc=p74;  # RAM address pin A16	
net we_n loc=p63; # RAM write-enable	
net oe_n loc=p62;	

© 2001 by XESS Corp.

```
net ce_n loc=p65; # RAM chip-enable
net s<0> loc=p21; # LED segment S0
net s<1> loc=p23; # LED segment S1
net s<2> loc=p19; # LED segment S2
net s<3> loc=p17; # LED segment S3
net s<4> loc=p18; # LED segment S4
net s<5> loc=p14; # LED segment S5
net s<6> loc=p15; # LED segment S6
```

#### Listing 7: Pin assignments for the XS40 Board.

<pre># pin assignments f</pre>	
	clock from programmable osc.
	reset from data pin D0 of parallel port
net d<0> loc=p41; #	
net d<1> loc=p40; #	
net d<2> loc=p40; #	
net d<3> loc=p38; #	-
net d<4> loc=p30; #	-
net d<5> loc=p81; #	-
net d<6> loc=p80; #	-
net d<7> loc=p10; #	-
<pre>net a&lt;0&gt; loc=p3; #</pre>	-
net a<1> loc=p4; #	
net a<2> loc=p5; #	
net a<3> loc=p78; #	
net a<4> loc=p79; #	
net a<5> loc=p82; #	
net a<6> loc=p83; #	RAM address pin A6
net a<7> loc=p84; #	RAM address pin A7
net a<8> loc=p59;#	RAM address pin A8
net a<9> loc=p57;#	-
±	# RAM address pin A10
	# RAM address pin All
	# RAM address pin A12
	# RAM address pin A13
	# RAM address pin A14
—	# RAM address pin A15
	# RAM address pin A16
<pre>net we_n loc=p62; #</pre>	
<pre>net oe_n loc=p61; #</pre>	
<pre>net ce_n loc=p65; #</pre>	-
net s<0> loc=p25; #	
net s<1> loc=p26; #	
net s<2> loc=p24; #	-
net s<3> loc=p20; #	-
net s<4> loc=p23; #	-
net s<5> loc=p18; #	-
net s<6> loc=p19; #	LED segment S6

Once the target device, top-level module, implementation options and constraint file are setup, click on the Run button to start the synthesis and implementation phases.

Synthesis/Impl	ementation	settin	gs	×
Top level:	ramsum		•	<u>R</u> un
Version name:	ver1			ОК
Synthesis Settin	gs:	9	SET	<u>C</u> ancel
				<u>H</u> elp
F Target Device				
Family: XC9	500	•		
Device: 9510	)8PC84	-	Speed:	-20 💌
Edit Synthesi	s/Implementa	tion cor	nstraints	
View Estimat	ed Performan	ce after	Optimiza	ation
🔽 Auto Run I	mplementation	n tools		
- Physical Imple				
Revis	ion name:	rev1		<u>O</u> ptions
Contro	ol Files:	SE	T	

Both phases should complete with no problems.

Project Manager 🛛 🔀
Flow Engine ver1->rev1 Completed Successfully.
ОК

#### Generating the Bitstream

Once the implementation phase is completed, we can go on to create the SVF file containing the configuration bitstream for the XC95108 CPLD.



Select the Output→Create SVF File... menu item from the **JTAG Programmer** window that appears.

🞇 dsgn5_1 - JTAG Programmer	_ 🗆 🗵
<u>File Edit Operations Output View Help</u>	
□ 🖙 🖬 🐰 📴 Cable Auto Connect 🚼 🖽 🛱 🖽 🤹 🎖 🕺	
Cable <u>R</u> eset	-
V Use Cable	
Create SVF File	
Append to SVF File	
XC95108	
dsgn5_1.jed	
TDO	
	<b>⊁</b>
Creates an SVF file and directs subsequent operations to it	

As always, specify an initial transition of the JTAG state machine through the Test-Logic-Reset state.

SVF Options		×
Initial transition to F	Run-Test/Idle:	
Through Test-L	.ogic-Reset	
C <u>S</u> kipping Test-I	_ogic-Reset	
ок	Cancel	<u>H</u> elp

Then tell the JTAG Programmer to save the configuration bitstream in the dsgn5\_1.svf file in the top-level directory of the **dsgn5\_1** project.

Create a New SVF File		? ×
Save in: 🗀 dsgn5_1	- 🗈 🜌	🗃 🔳
🗋 dpm_net		
🚊 dsgn5_1		
ib .		
xproj		
lagn5_1.svf		
File name: dsgn5_1.svf	 	<u>S</u> ave
Save as type: SVF Files(*.svf)	•	Cancel

Now initiate the generation of the bitstream by selecting the Operations  $\rightarrow$  Program menu item.

🏪 dsgn5j	1 - JTAG Programmer	
<u>F</u> ile <u>E</u> dit	Operations Output View Help	
	Program Verify	
	<u>E</u> rase	<b>▲</b>
	Eunctional Test	
	Blank Check	
	<u>R</u> eadback Jedec	
I I	Get Device <u>I</u> D	
TDI	Get Device Checksum	
	Get Device <u>S</u> ignature/Usercode	
	Chain <u>O</u> perations	
	dsgn5_1.jed	
TDO -		
		-
•		<u> </u>
Programs th	ne selected devices in the JTAG chain	SVF Mode

Just click on the OK button in the **Options** window to begin generating the bitstream.

Options	×
Program Options	
Erase Before Programmin	🖵 Skip user array
I ⊻erify	☐ <u>W</u> rite Protect
Eunctional Test	Read Protect
Parallel Modi	🗖 Load Fpga
External <u>P</u> in Verification	1 Pin #:
Usercode (8 Hex Chars	j FFFFFFF
ок 🖓	Cancel <u>H</u> elp

The bitstream generation should complete without incident.

Operation Status	×
Loading Boundary-Scan Description Language (BSDL) file 'C:/Fndtn/xc9500/data/xc95108.bsd'completed successfully. 'dsgn5_1(Device1)': Generating SVF vectors to check boundary-scan chain integritydone. 'dsgn5_1(Device1)': Generating SVF vectors to put device in ISP modedone. 'dsgn5_1(Device1)': Generating SVF vectors to erase devicedone. 'dsgn5_1(Device1)': Processing JEDEC filedone. 'dsgn5_1(Device1)': Generating SVF vectors to program devicedone. 'dsgn5_1(Device1)': SVF vector generation for programming completed successfully.	A F
All operations were completed successfully.	
OK View Log File	

#### Downloading and Testing the Design

We need some test data to store into the RAM of the XS95 Board in order to test the design. Go to the top-level directory of the *dsgn5\_1* project and use a text editor to create a file called data.hex containing this single line of text:

- OB 0000 FF FE FD FC FB FA F9 F8 F7 F6 F5  $\,$ 

This is a set of eleven data bytes that will be loaded into RAM starting at address zero. (This data is represented in the XESS format.) If you manually complement-and-sum these data values you will get the following result (in two-digit hexadecimal):

```
(-FF) + (-FE) + (-FD) + (-FC) + (-FB) + (-FA) + (-F9) + (-F8) + (-F7) + (-F6) + (-F5) =
(1) + (2) + (3) + (4) + (5) + (6) + (7) + (8) + (9) + (A) + (B) =
42
```

Now that the SVF and test data file are ready, connect an XS95 Board to the PC parallel port and start the GXSLOAD program. Go to the top-level directory for the **dsgn5\_1** project and select the dsgn5\_1.svf and data.hex files. Then drag-and-drop them into the **gxsload** window. The data file will be downloaded into the RAM and then the SVF file will be programmed into the XC95108 CPLD on the XS95 Board.

🔀 gxsload 📃 💌	🖿 dsgn5_1
Drop .BIT, .SVF, .HEX, and .EX0 files here to download to the XS or XSV Board. Recent Files:	File     Edit     View     Go     Favorites     Help       ↓     ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓       ↓     ↓     ↓     ↓     ↓
	Image: Second secon
Reload EEPROM Port LPT1	Image: Second

The reset for the circuit is controlled by data pin D0 of the parallel port. If D0 is at logic 1 after the downloading completes, the circuit will be held in the reset state and the LED will be blank. To release the reset, open the **gxsport** window and click on the D0 button until it displays a zero.



Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.

🔼 дхѕро	rt				_ 🗆 🗵
1 1 D7 D6 Strobe			 	D0	Exit
		Jank	 . jer i	·	

The answer is that you are probably running the design with a 50 MHz clock (the default for the XS95 Board). Can this design run that fast? Let's check the timing for the implemented design. Click on the icon for the report files in the **Project Navigator** window.

📣 dsgn5_1 - ver1 (95108-20PC)	84) - Project Manager	_ 🗆 ×	
<u>File D</u> ocument <u>View Project S</u>	ynthesis <u>I</u> mplementation <u>T</u> ools <u>H</u> elp		
Files Versions	Flow Contents Reports		
🖻 🗖 dsgn5_1		-	
🕀 🗳 leddod.vhd	ver1 rev1 (XC95108-20-PC84)		
🕀 🚭 ramsum.vhd			
- 🖯 dsgn5_1 - 🖯 simprims	🖺 📲 🌄 🏣 🛛		
⊖ xc9500			
	100 L		
	SYNTHESIS SIMULATION		
	IMPLEMENTATION VERIFICATION	-	
Pcm : Synopsys version: 3, 3, 1	0, 4517	-	
Pcm : Reading Synopsys/Xilin			
Console / HDL Errors / HDL Warnings / HDL Messages /			
	nings / HDL Messages /		
Ready			

Then double-click the Post Layout Timing Report in the **Report Browser** window.

🚯 Report Browser - dsgn5_1(ver1->rev1)			×
		Ď	
Translation Report	Fitting Report	Post Layout Timing Report	

The top portion of the timing report is shown in Listing 8 and this tells us what we want to know: the maximum clock frequency for this design is 5.5 MHz. The slow clock is brought about by the long carry propagation times through the complementors and adders in the design. The situation is made worse because the implementation algorithms have packed the logic to emphasize area efficiency and this can add extra propagation delays to the circuit.

#### Listing 8: Timing report for the design.

Performance Summary Report

Design: dsqn5 1 Device: XC95108-20-PC84 Program: Timing Report Generator: version C.22 Date: Sat Jan 05 14:08:28 2002 Performance Summary: Clock net 'clk' path delays: Clock Pad to Output Pad (tCO): 73.0ns (4 macrocell levels) Clock Pad 'clk' to Output Pad 's<4>' (GCK) 161.5ns (8 macrocell levels) Clock to Setup (tCYC): Clock to Q, net 'sum r<0>.Q' to TFF Setup(D) at 'sum r<7>.D' (GCK) Target FF drives output net 'sum r<7>' Setup to Clock at the Pad (tSU): 151.5ns (7 macrocell levels) Data signal 'd<0>' to TFF D input Pin at 'sum r<7>.D' Clock pad 'clk' (GCK) Minimum Clock Period: 161.5ns Maximum Internal Clock Speed: 6.1Mhz (Limited by Cycle Time)

We need to reduce the clock frequency of the XS95 Board to less that 6.1 MHz in order for our design to work reliably. To do this, start the GXSSETCLK program. Place 20 in the Divisor field to reduce the 100 MHz master frequency to 5 MHz. Then click on the SET button.

者 Set XS B	oard Clock Frequ	ency 📃 🗖 🗙
Board Type	XS95-108 💌	SET
Port	LPT1 💌	Exit
Divisor	20	External Clock
	oard clock frequency 100 MHz master free	

A set of instructions will appear that must be followed to adjust the clock frequency of the XS95 Board. After doing these steps, click on the OK button to reprogram the clock.



Reprogramming the clock takes a minute or two after which the following set of instructions is given to activate the new clock frequency.

GXSSET	GXSSETCLK 🛛 🕅					
•	The frequency of your XS95 Board has been set!! Now do these steps to activate the oscillator: 1) Remove the power and downloading cable from your XS95 Board 2) Move the shunt to the "osc" position of jumper J6 3) Reconnect the power cable 4) Reconnect the downloading cable 5) Click on the OK button					

# Using an Internal Synchronous RAM

The second version of the RAM summation circuit has the design hierarchy shown in Figure 15. The root module of the design sums the data stored in an internal synchronous RAM module while the LED decoder module displays the four-bit hexadecimal digits on a seven-segment display. Only the XC4000 FPGAs have internal RAM so this design can only be done using the XS40 Board. The XC95108 CPLD on the XS95 Board is not suitable for designs, which require large amounts of internal data storage.



Figure 15: Design hierarchy for a logic circuit that displays the summation of data in an internal synchronous RAM.

Each of these modules is stored in the *dsgn5\_2* directory that was created by starting an HDL project follows.

New Proje	X		
Name:	DSGN5_2		ок
Directory:	C:\PRAG211		Cancel
Туре:	F2.1i	•	<u>H</u> elp
Flow:	C <u>S</u> chematic	● HDL	

#### The Internal RAM Module

The first module we will add is the internal synchronous RAM. This module is constructed using the CORE Generator. To start this tool, select the Tools→Design Entry→Core Generator... menu item.

🐌 dsgn5_2 - design i	not implemented - Project Manag	er 💶 🗙
<u>File</u> <u>D</u> ocument <u>V</u> iew	Project Synthesis Implementation	<u>I</u> ools <u>H</u> elp
	S <u>c</u> hematic Editor	<u>D</u> esign Entry ►
	State E <u>d</u> itor	Simulation/Verification
Files Versions	<u>H</u> DL Editor	Implementation
□ □ dsgn5_2	Symbol <u>E</u> ditor	Device <u>P</u> rogramming ►
<sup>l</sup> <mark>⊖</mark> dsgn5_2	LogiBLOX module generator	Utilities
	<u>C</u> ORE Generator	
		-
	DESIGN ENTRY	Ŷ
	•	
	SYNTHESIS	SIMULATION
	-	
	影 (四)	1. to to
	IMPLEMENTATION	VERIFICATION
	PROGRAMMING	
		•
	\prag21i\dsgn5_2\ram.edn added rver initialization	<u> </u>
	\prag21i\dsgn5_2\ram.edn remov	ed 🗸
	/ HDL Warnings / HDL Messages /	
Run CoreGen		

The **Xilinx CORE Generator** window will appear. The left-hand pane of the window displays the various families of circuits that the tool can generate. The individual circuits within a highlighted family are shown in the right-hand pane.

Elle Project Corre Web Help     Project Path: C:\Prag21i\dsgn5_2 Target Family: SPARTAN     View mode: Taxonomy     Contents of:     Basic Elements   Communication & Networking   Digital Signal Processing   Math Functions   Math Functions   Math Functions   ProtoType & Development Hardware Products   Standard Bus Interfaces   Storage Elements & Memories
View mode: Taxonomy       Contents of:         Basic Elements       Name       Type       Version       Family         Digital Signal Processing       Math Functions       Microprocessors, Controllers & Peripherals       ProtoType & Development Hardware Products         Standard Bus Interfaces       Standard Bus Interfaces       Standard Bus Interfaces       Standard Bus Interfaces
View mode: Taxonomy       Contents of:         Image: Basic Elements       Name       Type       Version       Family         Image: Digital Signal Processing       Math Functions       Image: Basic Controllers & Peripherals       Image: Basic Controllers & Peripherals         Image: ProtoType & Development Hardware Products       Standard Bus Interfaces       Standard Bus Interfaces
Basic Elements       Name       Type       Version       Family         Digital Signal Processing       Microprocessors, Controllers & Peripherals       ProtoType & Development Hardware Products         Standard Bus Interfaces       Standard Bus Interfaces       Standard Bus Interfaces
Communication & Networking     Digital Signal Processing     Math Functions     Microprocessors, Controllers & Peripherals     ProtoType & Development Hardware Products     Standard Bus Interfaces
Digital Signal Processing     Math Functions     Microprocessors, Controllers & Peripherals     ProtoType & Development Hardware Products     Standard Bus Interfaces

Our first action is to setup the CORE Generator for the target FPGA and type of project we are using in Foundation. Click on the Project → Project Options... menu item to open the **Project Options** window.

🔆 Xilinx CORE Generator 2.1i				_ 🗆 🗵
<u>File Project C</u> ore <u>W</u> eb <u>H</u> elp				
C <u>N</u> ew				
Open				
	nily: SPARTAN			
View n Project Options	Contents of:			
🚞 Basic Elements	Name	Туре	Version	Family
Communication & Networking Digital Signal Processing				
Math Functions				
Microprocessors, Controllers & Peripherals				
ProtoType & Development Hardware Products     Standard Bus Interfaces				
Storage Elements & Memories				
	I			Þ
J				

In the **Project Options** window, select VHDL in the Design Entry section since our project will be done using VHDL. Also, click on the Synopsys button in the Vendor section since this is the VHDL synthesis tool used by Foundation 2.1. Finally, select XC4000 as the target FPGA in the Family section.

Project Options	
—Design Entry—	Vendor
C Schematic	C Foundation
VHDL	Synopsys
C Verilog	O Exemplar
J	O Synplicity
	O Other
	Netlist Bus Format 🛛 🖃 🗹
	Active Bus Format
Behavioral Simulati	on Family
VHDL	XC4000 💌
🔽 Verilog	XC4000
	Spartan <sup>V</sup>
	OK Spartan2

Once the options are set as described above, click on the OK button to close the window.

*	Project Options		×
Γ	Design Entry	Vendor	
	C Schematic	C Foundation	
	VHDL	Synopsys	
	C Verilog	C Exemplar	
		C Synplicity	
		C Other	
		Netlist Bus Format B <i></i>	
		Active Bus Format B <i></i>	
	Behavioral Simulatior	n Family <mark>XC4000 ▼</mark>	
	Verilog		
		OK Cancel	

In the **Core Generator** window we can now see the Target Family is listed as XC4000. Next, double-click on the Storage Elements & Memories entry in the left-hand pane.

🔆 Xilinx CORE Generator 2.1i						
<u>File Project Core Web H</u> elp						
🗋 🖻 🐇 🎆 🎇 🥕						
Project Path: CAPrag21ikdsgn5_2 Target Family: XC4000						
View mode: Taxonomy	Contents of: /Storage Elements & Memor	ies				
<ul> <li>Basic Elements</li> <li>Communication &amp; Networking</li> <li>Digital Signal Processing</li> <li>Math Functions</li> <li>Microprocessors, Controllers &amp; Peripherals</li> <li>ProtoType &amp; Development Hardware Products</li> <li>Standard Bus Interfaces</li> <li>Storage Elements &amp; Memories</li> </ul>	Name	Type Version	<u>Family</u>			
	<b>I</b>		Þ			

Double-clicking the Storage Elements & Memories entry expands it and exposes three subfamilies of modules. Clicking on the RAMs & ROMs entry will display the members of this family in the right-hand pane.

Kilinx CORE Generator 2.1i       Image: Core Web Help         File       Project Core       Web Help         Image: Section Core Web       Help         Image: Section Core Web						
View mode: Taxonomy  Basic Elements Communication & Networking Digital Signal Processing Math Functions Microprocessors, Controllers & Peripherals ProtoType & Development Hardware Products Standard Bus Interfaces Storage Elements & Memories Delay Elements Delay Elements Refined Streaments Communication Streaments Delay Elements Refined Streaments Refine	Contents of: /Storage Elements & Mem Name Dual Port Block Memory Registered DualPort RAM Registered ROM Registered SinglePort RAM Single Port Block Memory	Indiana State Stat	Family Virtex,Spartan2 XC4000,Spartan XC4000,Spartan XC4000,Spartan Virtex,Spartan2			

We need to both read and write the data values so a RAM should be used rather than a ROM for this application. We will also try to keep this circuit as similar to the one in the previous project so we will use a RAM with a single data port for both read and write operations. For these reasons, the Registered SinglePort RAM is the closest match to what we need so double-click that entry to begin the generation of such a RAM module.

Image: Second state in the second s						
View mode: Taxonomy	Contents of: /Storage Elements & Mem	ories/RAMs & ROMs				
Basic Elements	Name	Type Version	Family			
Communication & Networking	Dual Port Block Memory	<b>logi<sup>(C)</sup>PF</b> 1.0	Virtex,Spartan2			
Digital Signal Processing	Registered DualPort RAM	logic PE 1.0	XC4000,Spartan			
Microprocessors, Controllers & Peripherals	Registered ROM	logiciere 1.0	XC4000,Spartan			
ProtoType & Development Hardware Products	Registered SinglePort RAM Single Port Block Memory	1.0 IogiCQPE 1.0	XC4000,Spartan Virtex,Spartan2			
Storage Elements & Memories  Comparison  Comparison	<b>I</b>		Þ			
Registered SinglePort RAM						
The **Registered SinglePort RAM** window that appears has three tabs. The Core Overview tab displays a general summary of the module while the Contact tab lists the organization that was responsible for designing the module. But the Parameters tab is where we actually personalize the module to fit our particular application.

📸 Registered SinglePort RAM	x
📓 Parameters 📓 Core Overview 📓 Contact	1
	Registered SinglePort RAM
a q d d we	Component Name: Depth: 32 💌 Address Width: 5 Data Width: 8 💌
	Load Init Values       Show Init Values         .coe file:       no init values read - default is all 0s
CenerateCancel	Data Sheet

The first thing to do is to type a name for the module into the Component Name field. We chose the very original name RAM in this case.

Registered SinglePort RAM	ect
a q d we c c ce	Registered SinglePort RAM   Component Name:   RAM   Depth:   32   Address Width:   5   Data Width:   8   Load Init Values Show Init Valuescoe file: no init values read - default is all 0s
<u>G</u> enerate <u>C</u> ancel	Data Sheet

© 2001 by XESS Corp.

Next, we set the number of locations in the RAM module. We want to sum as many as sixteen values, so select 16 from the Depth pull-down menu.

👹 Registered SinglePort RAM		×
🎇 Parameters 🔀 Core Overview	Contact	1
	Registered SinglePort RAM	
	Component Name: RAM	
a q	Depth: 32 💌 Address Width: 5	
we c	Data Width:	
ce	Load Init Va 80 Bhow Init Values	
	.coe file: no init 96 112 d - default is all 0s	
	128 144 160	
	176 192	
<u>G</u> enerate <u>C</u> ancel		

Once the Depth field is set to sixteen, note that the Address Width field changes to four. The Data Width Field is already set to eight so there is no need to change it.

The RAM has to be initialized with the values that will be summed. In the previous example, this initialization was managed by having the GXSLOAD utility load the external RAM with the contents of a HEX file. But in this example, the RAM is contained within the FPGA so there is no way for GXSLOAD to access it and load its contents. Instead, the initial values for the RAM must be inserted into the FPGA configuration bitstream so the RAM contents are initialized at the same time the logic gates on the FPGA are configured. The Core Generator looks for RAM initialization values in .coe files. The contents of such a file for the RAM in this example is shown in Listing 9. The Radix field is set to sixteen to indicate the data is represented in hexadecimal form. The memdata field stores the initial values of each RAM location starting from address zero and incrementing upwards until all sixteen locations are filled.

### Listing 9: Initialization file for a Core Generator RAM.

```
Component_Name=ram;
Data_Width = 8;
Address_Width = 4;
Depth = 16;
Radix = 16;
memdata=FF,FE,FD,FC,FB,FA,F9,F8,F7,F6,F5,F4,F3,F2,F1,F0;
```

The RAM initialization values are stored in a file called ram.coe in the top-level directory of the *dsgn5\_2* project. To load these values into the Core Generator, click on the Load Init values... button as shown below.

Begistered SinglePort RAM	×
Parameters Core Overview Contact	Registered SinglePort RAM         Component Name:       RAM         Depth:       16       Address Width:       4         Data Width:       8       •       •         Load Init Values.       Show Init Values       •         .coe file:       no init values read - default is all 0s       •
<u>G</u> enerate <u>C</u> ancel	Data Sheet

Next, highlight the ram.coe file in the **Select coe file...** window and click on the Open button. This loads the RAM initialization values into the Core Generator.

Select coe fi	e			? ×
Look jn: 🔁	dsgn5_2	- 🗈	<u> 1</u>	* 🔳
🗀 dpm_net				
alsgn5_2				
lib				
ram.coe				
File <u>n</u> ame:	ram.coe			<u>O</u> pen
Files of <u>type</u> :	All Files (*.*)		নি	Cancel
2.	J			

Once the initialization values are loaded, click on the Show Init Values... button to view them.

😹 Registered SinglePort RAM	×
Parameters Core Overview Con	Registered SinglePort RAM         Component Name:       ram         Depth:       16         Depth:       16         Data Width:       8         Load Init Values       Show Init Values         .coe file:       C:\Prag21i\dsgn5_2\ram.coe
<u>G</u> enerate <u>C</u> ancel	Data Sheet

The initial value for each RAM location will appear in the **Coefficients** window. (The locations are labeled Coef# because RAMs inside FPGAs are often used to store tables of coefficients for digital signal processing applications.) Click on the Close button to remove the window.

👸 Coefficients			×			
all coefficient v	values are show	/n as HEX				
Coef#0 = ff	Coef#1 = fe	Coef#2 = fd	Coef#3 = fc			
Coef#4 = fb	Coef#5 = fa	Coef#6 = f9	Coef#7 = f8			
Coef#8 = f7	Coef#9 = f6	Coef#10 = f5	Coef#11 = f4			
Coef#12 = f3	Coef#13 = f2	Coef#14 = f1	Coef#15 = f0			

Once the RAM width, depth and initial values are specified, click the Generate button to have Core Generator assemble the necessary files that describe this RAM.

Registered SinglePort RAM	Contact Registered SinglePort RAM Component Name: ram Depth: 16 Address Width: 4 Data Width: 8 Load Init Values Show Init Values .coe file: C:\Prag211idsgn5_2\ram.coe
<u>Generate</u> Generate	Data Sheet

The success of the operation will be reported in the bottom pane of the **Core Generator** window.

Xilinx CORE Generator 2.1i			
<u>F</u> ile <u>P</u> roject <u>C</u> ore <u>W</u> eb <u>H</u> elp			
🗋 🖻 🐐 📰 🐰 🥕			
Project Path: C:\Prag21i\dsgn5_2  Target Far	nily: XC4000		
View mode: Taxonomy	Contents of: /Storage Elements & Mem	nories/RAMs & ROMs	
🧰 Basic Elements	Name	Type Version	Family
Communication & Networking     Digital Signal Processing     Math Functions	Dual Port Block Memory Registered DualPort RAM Registered ROM	logiC <sup>ARE</sup> 1.0 logiC <sup>ARE</sup> 1.0 logiC <sup>ARE</sup> 1.0	Virtex,Spartan2 XC4000,Spartan XC4000,Spartan
Microprocessors, Controllers & Peripherals	Registered SinglePort RAM	1.0	XC4000,Spartan
ProtoType & Development Hardware Products Standard Bus Interfaces Storage Elements & Memories Delay Elements Delay Elements Refine FIFOS RAMS & ROMS	Single Port Block Memory	Logi CAPE 1.0	Virtex,Spartan2
Successfully generated ram (Registered_Singlef	Port_RAM 1.0)		

dsgn5\_2 - design not implemented - Project Manager - 🗆 × File Document View Project Synthesis Implementation Tools Help Add.. ٦ 🗭 🖻 主 🔈 💦 2h. Remove 😽 Del Fil Contents \[ Reports <u>O</u>pen Enter Info... Ctrl+I dsgn5\_2 Expand One Level + Expand Branch Expand All Ctrl+\* Collapse Branch -DESIGN ENTRY Find Object... HDL П SYNTHESIS SIMULATION 19 ž Π VERIFICATION IMPLEMENTATION 💡 9 -Ę, PROGRAMMING -Xilinx CORE Generator is starting ٠ Conv : C:\PRAG21I\DSGN5\_2\RAM.xsf Conv : Import netlist completed in 0.1 [s]. Console / HDL Errors / HDL Warnings / HDL Messages . Add existing document to project

Once the RAM module is generated, we can add it to the project using the Document  $\rightarrow$  Add... menu command.

The Core Generator creates modules in the form of EDIF netlist files with the .edn extension. Select Edif Sources in the Files of type field of the **Add Document** window and then you will see the ram.edn file in the top-level directory of the project. Highlight ram.edn and click on the Open button to add the RAM module to the project.

Add Docume	nt					? ×
Look jn: 🔂	dsgn5_2	-	£		<b>C</b>	<b></b>
dsgn5_2						
🚞 xproj						
🗐 ram.edn						
L					_	
File <u>n</u> ame:	ram.edn					<u>Open</u>
Files of <u>type</u> :	Edif Sources (*.EDF;*.EDN;*.ED	0)		•		Cancel
						<u>H</u> elp

🔹 dsgn5_2 - design not implemented - Project Manager	x
<u>File D</u> ocument <u>View Project</u> <u>Synthesis</u> <u>Implementation</u> <u>T</u> ools <u>H</u> elp	
Files Version:  Flow Contents Reports	Τ
□	-
+ 🔽 <mark>ram.edn</mark> dsgn5_2	
└── 🖯 dsgn5_2	
📄 🔛 🥵 🏣-	
DESIGN ENTRY	
🖺 🔹 🦫 👘 😓	
20. 00 · · · · · · · · · · · · · · · · ·	
IMPLEMENTATION ? VERIFICATION	
PROGRAMMING	-
Dpm : Done	-
Pcm : Document c:\prag21i\dsgn5_2\ram.edn added	
Pcm : Synopsys server initialization Console / HDL Errors / HDL Warnings / HDL Messages /	4
Ready	4

After adding the RAM module, the **Project Navigator** window appears as shown below.

## The LED Decoder Module

This LED decoder circuit for this project is identical to the one used in the previous project. Use the Document  $\rightarrow$  Add... menu command and then move to the top-level directory of dsgn5\_1. Highlight the leddcd.vhd file and click on the Open button to add the RAM module to the project.

Add Docume	nt						? ×
Look jn: 🔁	) dsgn5_1	•	£	<u></u>	Ě		#
dpm_net dsgn5_1 lib xproj eddcd.vh							
File <u>n</u> ame:	leddcd.vhd					<u>O</u> pen	Ņ
Files of type:	HDL (*.VHD;*.VER;*.VE;*.V)			•		Cancel <u>H</u> elp	

After adding the LED decoder module, the **Project Navigator** window appears as shown below.

🕪 dsgn5_2 - design not implemented - Project Manager	_ 🗆 ×
<u>File D</u> ocument <u>View</u> <u>Project</u> <u>Synthesis</u> <u>Implementation</u> <u>T</u> ools <u>H</u> elp	
▶ 🕞 📵 🐽 🗢 😰 🗭 🖻 🖶 💦	
Files     Versions     Flow     Contents     Reports       Image: Second s	
+ @/leddcd.vhd G dsgn5_2 DESIGN ENTRY	
SYNTHESIS SYNTHESIS	
Implementation ?	
PROGRAMMING	<b>_</b>
Dpm : Done	<b></b>
Pcm : Document c:\prag21i\dsgn5_2\leddcd.vhd added Pcm : Synopsys server initialization	-
Console / HDL Errors / HDL Warnings / HDL Messages /	
Ready	

## The Root Module

The root module sequences through three main phases just as in the previous example:

- **Phase 1:** Starting from an upper address of RAM and proceeding to address zero, the value stored at each RAM address is read and the two's-complement is computed and written back to the same address.
- **Phase 2:** Restarting from the upper address and proceeding to address zero, each value is read from RAM and added to a sum register.
- **Phase 3:** The sum is displayed on the seven-segment LED by blanking the LED segments for a long interval to signal the start of the sum, then the hexadecimal digit for the upper four bits of the sum are displayed, then the LEDs are blanked for a shorter interval and then the hexadecimal digit for the lower four bits is displayed. Then this four-step display process repeats.

The VHDL code for the root module (Listing 10) was derived from the root module of the previous example is in the ramsum.vhd. The differences between the previous root module and this one are described below.

- Lines 6–12: The RAM address, data and control signals are no longer included in the interface definition. That's because the RAM is now internal to the FPGA so we don't need any I/O pins to interface to the external RAM chip.
- Lines 15–24: These lines define the interface to the RAM module created by the Core Generator. In addition to the ram.edn netlist file, the Core Generator also creates a ram.vho file that shows the VHDL interface definition for the component and how to instantiate it. We just copied the component declaration from that file.
- Lines 25–32: The internal buses for interfacing to the RAM module are declared on these lines as well as the four-bit address register. The address, input data and output data buses used in the state machine are declared with the type UNSIGNED. This makes it easier to perform arithmetic operations on their values using the numeric\_std library. But the RAM module from the Core Generator has input and output buses declared as type STD\_LOGIC\_VECTOR so some intermediary buses are declared to make the type conversion.
- Line 33: This line declares the constant for the address of the upper end of the RAM data range that will be summed. As in the previous example, this circuit will complement and sum eleven bytes of data from address zero to ten, inclusive.
- **Lines 34:** A register to hold the sum of the RAM values is declared here. The register to hold the current value read from the RAM in the previous example is no longer needed here because the synchronous RAM outputs will remain stable except on the rising edge of the clock.
- Lines 42–43: Only eight states are defined for this design. The invertnop state is no longer needed when a synchronous RAM is used.
- Lines 57–58: The default values for the active-high RAM read-enable (ce) and writeenable (we) signals are defined here. The logic 1 on the ce input means a rising clock edge will cause the RAM to register the value of the currently addressed location onto its outputs. The logic 0 on the we input disables any writes to the RAM.
- Line 59: The input data bus to the RAM is set to zero unless it is specifically set to some other value in the state machine. Unlike the example with the external RAM, the input and output data buses of the internal RAM module are separate so we do not need to tristate the bus when it is not in use.
- Lines 62–64: The init state initializes the state machine for the start of the loop that complements the contents of RAM. The address register is set to point to the upper bound of the RAM data range and the state machine is moved to the start of the two's-complement loop (invertw).

- **Lines 65–68:** The invertw state activates the write-enable of the RAM. At the time this state is entered, the contents of the RAM address generated in the previous cycle will be available on the RAM output data bus. This value is complemented and written back on the din bus to the same address location. The actual write will take place on the next rising-edge of the clock. Then the state machine is moved to the invertr state to read the next RAM location.
- **Lines 69–78:** The invertr state determines the next location to be read from RAM depending upon the value of the current address register. If the current RAM address has reached zero, then the address register is reloaded with the starting address of the data range and control branches to the add state where the summation of the data takes place. Otherwise, the current address is decremented and control returns to the invertw state so the next data location can be complemented. In either case, the contents at the new address will be available on the outputs of the RAM at the start of the next clock cycle.
- Lines 79–88: The add state adds the value from RAM to the summation register. If the current RAM address is zero indicating the summation loop is finished, then the time delay register is loaded with the initial blanking interval for the LED display. Then the state machine is moved to the start of the display loop (display blank). If all the RAM data has not been summed, then the RAM address is decremented and the state machine stays in the add state. The contents at the new address will be available on the outputs of the RAM at the start of the next clock cycle.
- Lines 139–149: The UNSIGNED address and data buses used in the state machine are converted to the STD LOGIC VECTOR types and passed to the RAM module created using the Core Generator. As with the component declaration, example code for instantiating the RAM module can be found in the ram.vho file

```
Listing 10: VHDL code for the root module.
 1
     library IEEE;
 2
     use IEEE.std logic 1164.all;
 3
     use IEEE.numeric std.all;
 4
     use WORK.leddcd pckg.all;
-5
6
7
     entity ramsum is
       port (
 8
          rst : in STD LOGIC;
                                                       -- reset
 ğ
           clk : in STD LOGIC;
                                                       -- clock
10
               : out STD LOGIC VECTOR(6 downto 0) -- outputs to LED segments
          S
11
        );
12
     end ramsum;
13
14
     architecture ramsum arch of ramsum is
15
     component ram -- 16-byte synchronous RAM from CoreGen
16
        port (
           a : IN std_logic_VECTOR(3 downto 0); -- address bus
17
               : IN std_logic_VECTOR(7 downto 0); -- data input bus
: IN std_logic; -- write-enable
18
           d
19
           we
                                                        -- write-enable
                : IN std logic;
20
                                                        -- clock
          С
           ce : IN std_logic;
21
                                                        -- read-enable
22
                : OUT std logic VECTOR(7 downto 0) -- data output bus
           q
23
        );
24
     end component;
     © 2001 by XESS Corp.
```

```
25
     -- RAM address, data, control signals
26
27
28
29
30
31
32
33
35
     signal addr r, next addr : UNSIGNED(3 downto 0); -- RAM address reg
             din : UNSIGNED(7 downto 0); -- RAM data input bus
     signal
             dout
                                                     -- RAM data output bus
     signal
                       : UNSIGNED(7 downto 0);
     signal
              ce : STD_LOGIC;
                                                     -- RAM chip-enable
              we : STD_LOGIC; -- RAM write-enable
aa : STD_LOGIC_VECTOR(addr_r'range); -- RAM address bus
dd,qq: STD_LOGIC_VECTOR(din'range); -- RAM data I/O buses
     signal
     signal
      signal
     constantmaxaddr : UNSIGNED := TO UNSIGNED(10,addr r'length);
              sum r, next sum : UNSIGNED(din'length-1 downto 0); -- RAM sum
     signal
               delay_r, next_delay : UNSIGNED(22 downto 0); -- delay counter
     signal
36
     constantblank dly : UNSIGNED := TO_UNSIGNED(5_000_000,delay_r'length);
37
     constant interdigit dly:UNSIGNED:=TO UNSIGNED(1_600_000, delay_r'length);
38
     constantdigit_dly : UNSIGNED := TO_UNSIGNED(2_500_000,delay_r'length);
39
              digit: UNSIGNED(3 downto 0); -- LED hex digit to display
     signal
40
                                            -- LED digit blanking signal
               blank: STD LOGIC;
     signal
41
     -- states for the state machine
42
     type state is (init, invertr, invertw, add, display blank,
43
              display_upper_digit,display_interdigit,display_lower_digit);
44
              st r, next st: state; -- state register
     signal
45
     begin
46
47
      -- this process computes the actions of the state machine in each state
48
     process(clk,st r,addr r,sum r,delay r,din)
49
     begin
50
       -- default outputs unless otherwise specified
51
52
                    <= st r; -- remain in the current state
        next st
        next_addr <= addr r;
                                  -- don't change the RAM address
53
        next sum <= sum r; -- don't update the sum register
54
        next delay <= delay r-1; -- decrement the delay counter
55
56
57
58
59
        digit <= TO UNSIGNED(0,digit'length); -- output a '0' LED digit</pre>
                   <= '1'; -- blank the LED display
        blank
                     <= '1'; -- always read the RAM
<= '0'; -- dop't ......</pre>
                   <= '1';
        се
                                      -- don't write to the RAM
        we
        din
                      <= TO UNSIGNED(0, din'length);
60
61
        case st r is -- case statement for the state machine
62
        when init => -- initialization state
63
          next_addr <= maxaddr; -- start inverting from the upper address
next_st <= invertw; -- enter the RAM inversion loop</pre>
64
65
        when invertw => -- write inverted byte value into same \bar{RAM} location
66
           we <= '1'; -- write RAM at the end of this clock cycle
67
           din <= TO UNSIGNED(0,din'length) - dout;-- output inverted byte</pre>
68
           next st <= invertr; -- now read from next RAM location</pre>
69
        when invertr => -- read byte from RAM
70
71
72
73
74
75
76
77
78
79
80
           if addr r = TO UNSIGNED(0, addr r'length) then
              -- reached the lower address of the RAM data
             next_addr <= maxaddr; -- reload register with upper address</pre>
             next st <= add;</pre>
                                       -- enter the summation loop
           else
              -- haven't inverted all the RAM data yet
              next addr <= addr r - 1;</pre>
              next st <= invertw;</pre>
                                        -- now write to it
           end if;
                        -- sum the inverted data from RAM
        when add =>
           next sum <= sum r + dout; -- add the RAM data to the sum
81
           if addr r = TO UNSIGNED(0, addr r'length) then
82
              -- reached the lower address of the RAM data
83
              next delay <= blank dly; -- load display interval counter
84
              next st <= display blank; -- now display the sum
85
           else
86
              -- haven't summed all the RAM data yet so stay in this state
87
              next addr <= addr_r - 1;-- address the next RAM location</pre>
88
           end if;
89
        when display blank => -- blank the display
```

© 2001 by XESS Corp.

```
90
            if delay r = TO UNSIGNED(0, delay r'length) then
 91
               -- initial display blanking is complete
 92
               next delay <= digit dly;</pre>
                                            -- load digit display interval
 93
               next st <= display upper digit; -- display the upper sum digit
 94
            end if;
 95
         when display upper digit => -- display the upper digit of the sum
 96
            blank <= '0';
                                           -- activate the LED
 97
            digit <= sum r(7 downto 4); -- display the upper 4-bits of the sum
 98
            if delay r = TO UNSIGNED(0, delay r'length) then
 99
               -- upper digit display is complete
100
               next_delay <= interdigit_dly; -- load inter-digit blank interval</pre>
               next_st <= display_interdigit; -- blank display between digits</pre>
101
102
            end if;
103
          when display interdigit => -- blank the display between sum digits
104
            if delay r = TO UNSIGNED(0, delay r'length) then
105
               -- inter-digit display blanking is complete
106
               next delay <= digit dly;</pre>
                                           -- load digit display interval
107
               next st <= display lower digit; -- display the lower sum digit
108
             end if;
109
         when display lower digit => -- display the lower digit of the sum
                                            -- activate the LED
110
            blank <= '0';</pre>
            digit <= sum r(3 downto 0); -- display the lower 4-bits of the sum
111
112
            if delay r = TO UNSIGNED(0, delay r'length) then
113
               -- lower digit display is complete
114
               next delay <= blank dly; -- load blank interval between loops
115
               next st <= display blank; -- loop and display the sum again
116
            end if:
117
         when others => -- error state
118
            next st <= init;-- re-initialize the state machine</pre>
119
         end case;
120
       end process;
121
122
       -- this process updates the registers on every rising clock edge
123
       process(clk)
124
125
126
       begin
          if clk'event and clk='1' then -- trigger on rising clock edge
            if rst='1' then -- synchronous reset
  st_r <= init;</pre>
127
128
               sum r <= TO UNSIGNED(0, sum r'length);</pre>
129
            else
                                  -- update the registers
130
                     <= next_st;
              st r
131
               sum r <= next sum;
132
              addr r <= next addr;
133
               delay r <= next delay;</pre>
134
            end if;
135
          end if;
136
       end process;
137
138
       -- connect clock, address, data and control to the RAM block
      aa <= STD_LOGIC_VECTOR(next_addr);
dd <= STD_LOGIC_VECTOR(din);
qq <= STD_LOGIC_VECTOR(dout);</pre>
139
140
141
142
       u2: ram port map (
143
               a => aa,
144
               d \Rightarrow dd,
145
               we => we,
146
               c \Rightarrow clk
147
               ce => ce,
148
               q => qq
149
               );
150
151
       -- output digit on the LED display
152
       u1: leddcd port map(blank=>blank, d=>digit, s=>s);
153
154
       end ramsum arch;
```

```
© 2001 by XESS Corp.
```

Figure 16 shows the waveforms for the last few cycles of the RAM complementation loop and the first few cycles of the summation loop. Data from RAM address 1 is available at the start of the invertw clock cycle. The data is complemented and sent back to the RAM where it is written at the end of the invertw cycle. During the following invertr cycle, the RAM address is decremented to zero and this is output on the RAM address bus. At the beginning of the next invertw cycle, the contents of RAM address 0 become available on the RAM data outputs. The current RAM address of zero is also stored in the address register in the FPGA. The complemented contents of RAM address 0 are written back into the RAM and control returns to the invertr state. Since the address register now contains zero, the RAM address is restored back to the start of the data range and control proceeds to the add state. During the add state the data from the address for the next RAM location is sent to the RAM. The data at the decremented address is available during the next clock cycle and the summation continues until the address register reaches zero.



Figure 16: Timing waveforms for the synchronous RAM summation circuit.

The timing waveforms illustrate the fundamental principles involved when writing to a synchronous RAM:

- The address, data and write-enable signal must be held stable for the setup time before the actual write-operation occurs at the next rising clock edge. Changing the address while the write-enable is active and the clock is either high or low will not cause erroneous writes into other addresses because write operations only occur on a rising clock edge.
- 2. There is no need to hold the address or data stable after the rising clock edge during a write operation.

3. For a synchronous RAM with registered outputs, the RAM outputs will show the data that was in the RAM location whose address was present at the previous rising clock edge. These outputs will persist until the next rising clock edge.

For our design, note that the RAM address, data and write-enable are stable before a rising clock edge and then change immediately after the edge. The output RAM data is stable for the entire cycle after the rising clock edge even when the RAM address changes. That means our design can complement the RAM data directly and then send it back to the RAM rather than store it in a register and then operate on it.

# Synthesizing and Implementing the Design

Once the modules are checked for syntax and any errors are removed, we can run the synthesis and implementation tools to create the configuration bitstream for the FPGA or CPLD. Click on the Implementation icon to run the synthesizer and the implementation tools sequentially.



We will target this design to the XS40 Board, so set the target device to be an XC4005XLPC84 with a -3 speed grade. Then select the **ramsum** module as the top-

level module for the design. Then click on the SET button so we can specify the constraint file containing the pin assignments.

Synthesis/Implementation settings	×
Top level: ramsum	<u>R</u> un
Version name: ver1	OK
Synthesis Settings: SET	<u>C</u> ancel
	<u>H</u> elp
Target Device	
Family: XC4000XL 💌	
Device: 4005XLPC84 Speed:	xl-3
Edit Synthesis/Implementation constraints	
F View Estimated Performance after Optimizat	tion
Auto Run Implementation tools	
Physical Implementation settings	
	Online 1
Revision name: rev1	<u>O</u> ptions
Control Files: SET	

Select the Custom entry in the drop-down list of constraint files.

Settings	×
Synthesis Settings	Implementation control files
Current Revision Control F	ile Settings:
Use Constraints file from:	None
Copy Guide file from:	Custom
Copy Floorplan files from:	None
Current Revision Control fil	e use:
🗖 Enable Gui	ded MAP and PAR
🗖 Enable Floo	orplanning
OK Cancel	<u>H</u> elp

The **Custom** window should appear with the dsgn5\_2.ucf file already in the Constraints File field. If not, click on the Browse button, find this file in the top-level directory of the **dsgn5\_2** project and select it. Then click on the OK button.

Custom	×
<u>C</u> onstraints File: dsgn5_2.ucf	<u>B</u> rowse
<u> </u>	<u>H</u> elp

The dsgn5\_2.ucf file should specify the assignments for the FPGA or CPLD pins that connect to the clock, reset, seven-segment LED and RAM address, data and control pins as shown in Figure 17. The pin assignments for the XS40 Board (which is our target for this example) are shown in Listing 11.



Figure 17: Connection of the programmable oscillator, parallel port, and LED digit to the pins of the FPGA or CPLD on the XS40 Board.

### Listing 11: Pin assignments for the XS40 Board.

<pre># pin assignments for XS40 Board</pre>
<pre>net clk loc=p13; # clock from programmable osc.</pre>
<pre>net rst loc=p44; # reset from data pin D0 of parallel port</pre>
net s<0> loc=p25; # LED segment S0
net s<1> loc=p26; # LED segment S1
net s<2> loc=p24; # LED segment S2
net s<3> loc=p20; # LED segment S3
net s<4> loc=p23; # LED segment S4
net s<5> loc=p18; # LED segment S5
net s<6> loc=p19; # LED segment S6

Once the target device, top-level module, implementation options and constraint file are setup, click on the Run button to start the synthesis and implementation phases.

Synthesis/Imp	ementatio	n setting	js	×
Top level: Version name:	ramsum ver1		•	<u>B</u> un OK
Synthesis Settin	gs:	S	ET	<u>C</u> ancel
Target Device Family: XC4 Device: 4005	000XL	• •	Speed:	Help
Edit Synthes	•			tion
Physical Imple				
Revis	ion name:	rev1		<u>O</u> ptions
Contre	ol Files:	SE	<b>T</b>	

Both phases should complete with no problems.

Project Manager 🛛 🕅
Flow Engine ver1->rev1 Completed Successfully.
ОК

## Downloading and Testing the Design

The bitstream file in this example contains both the FPGA configuration and the initial data for the internal RAM, so there is no need to create a separate data file to initialize the RAM as in the last example. The data in the internal RAM is identical to what was used in the previous example, so the result of the complement-and-sum process should be the same: 42 in two-digit hexadecimal.

Connect an XS40 Board to the PC parallel port and start the GXSLOAD program. Go to the top-level directory for the *dsgn5\_2* project and select the dsgn5\_2.bit file. Then dragand-drop it into the **gxsload** window. The bitstream file will be programmed into the XC4005XL FPGA on the XS40 Board.

🛃 gxsload 📃 🗖 🗙	🔁 dsgn5_2
Drop .BIT, .SVF, .HEX, and .EX0 files here to download to the XS or XSV Board. Recent Files:	File       Edit       View       Go       Favorites       Help         ↓       ↓       ↓       ↓       ↓       ↓       ↓         ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓         ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓       ↓         ↓
	dpm_net       coregen.prj       ld_1.xnf         dsgn5_2       dsgn5_2.alb       leddcd.er         lib       dsgn5_2.blt       leddcd.log         xproj       dsgn5_2.ll       leddcd.vhd         core.bat       dsgn5_2.prj       logiblox.ini         ib       dsgn5_2.prj       newcore.cm
Elecad EEPROM Port LPT1	coregen.fin dsgn5_2.xnf ram.coe coregen.log ram.edn  coregen.log dsgn5_2.xnf

The reset for the circuit is controlled by data pin D0 of the parallel port. If D0 is at logic 1 after the downloading completes, the circuit will be held in the reset state and the LED will be blank. To release the reset, open the **gxsport** window and click on the D0 button until it displays a zero.

🔼 gxsport					_ 🗆 🗵
1 1 D7 D6 <u>S</u> trobe	1 D5	D4 D3	1 1 D2 D1 Port LP	DOM	Exit

Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.

🗶 gxsport	
1 1 1 1 1	1 1 0 Exit
D7 D6 D5 D4 D3	D2 D1 D0
Strobe □ Count	Port LPT1 •

The answer is that you are probably running the design with a 50 MHz clock (the default for the XS40 Board). But the constants that determine the blanking and display intervals for the LED digit were calculated based on a 5 MHz clock. Can this design even run at 50 MHz? Let's check the timing for the implemented design. Click on the icon for the report files in the **Project Navigator** window.



Then double-click the Post Layout Timing Report in the Report Browser window.

🚯 Report Browser - dsgn5_2(ver1->rev1)				×
Translation Report	Map Report	Place & Route Report	Pad Report	Asynchronous Delay Report
Post Layout Timing Report	Bitgen Report			

The top portion of the timing report is shown in Listing 12 and this tells us what we want to know: the minimum clock period for this design is 40.096 ns which translates to a

maximum operating frequency of 24.94 MHz. The clock frequency is higher for this design than in the last example because the XC4005XL FPGA has specialized carry propagation circuitry that speeds the addition and complementation operations. It is not surprising that this design runs at 50 MHz when the FPGA is at room temperature and the power supply is optimal.

```
Listing 12: Timing report for the design.
```

```
_____
Xilinx TRACE, Version C.22
Copyright (c) 1995-1999 Xilinx, Inc. All rights reserved.
Design file:
                 dsgn5 2.ncd
Physical constraint file: dsgn5_2.pcf
Device, speed: xc4005xl,-3 (C 1.1.2.2 FINAL)
Report level: error report
WARNING: Timing: 181 - No timing constraints found, doing default enumeration.
_____
Timing constraint: Default period analysis
3080 items analyzed, 0 timing errors detected.
Minimum period is 40.096ns.
  _____
_____
Timing constraint: Default net enumeration
156 items analyzed, 0 timing errors detected.
Maximum net delay is 13.074ns.
_____
•••
...
Timing summary:
_____
Timing errors: 0 Score: 0
Constraints cover 3080 paths, 156 nets, and 525 connections (100.0% coverage)
Design statistics:
  Minimum period: 40.096ns (Maximum frequency: 24.940MHz)
  Maximum net delay: 13.074ns
Analysis completed Wed Jan 30 16:45:43 2002
_____
```

We need to reduce the clock frequency of the XS40 Board to 5 MHz to slow the display of the sum. To do this, start the GXSSETCLK program. Set the Board Type field to XS40-005XL. Place 20 in the Divisor field to reduce the 100 MHz master frequency to 5 MHz. Then click on the SET button.

<mark>X</mark> Set XS B	oard Clock Frequ	ency 📃 🗖 🗙		
Board Type	XS40-005XL 💌	SET		
Port	LPT1 💌	Exit		
Divisor	20	External Clock		
Set the XS Board clock frequency by entering a divisor for the 100 MHz master frequency				

A set of instructions will appear that must be followed to adjust the clock frequency of the XS40 Board. After doing these steps, click on the OK button to reprogram the clock.

GXSSETCLK	
٩	Before setting the XS40 Board frequency you must: 1) Remove the power and downloading cable from your XS40 Board 2) Place a shunt on the "set" position of jumper J12 3) Reconnect the downloading cable 4) Reconnect the power cable 5) Click on the OK button
	Cancel

Reprogramming the clock takes less than a minute after which the following set of instructions is given to activate the new clock frequency.

