Synthesizable VHDL Design for FPGAs

Eduardo Augusto Bezerra
Djones Vinicius Lettnin

Universidade Federal de Santa Catarina
Florianópolis, Brazil

August 2013

Contents

Chapter 2. HDL Based Designs

In this chapter the FPGA design flow is revisited, but using a hardware description language for the design entry instead of the schematic diagram discussed in Chapter 1. At the end of the chapter, the students should be able:

- to understand the basic structure of a VHDL description;
- to model a simple digital circuit in VHDL;
- to follow the VHDL design flow;
- to simulate and test a digital circuit designed in VHDL;
- to prototype a circuit described in VHDL, using an FPGA board.

2.1 Theoretical Background

A Hardware Description Language (HDL) is a modeling language used to describe the structure and behavior of a digital circuit. In addition, an HDL allows the test of the designed circuits through simulations. Hardware description languages express a temporal behavior and circuit structure in a text format. Additionally, the syntax and semantics of HDLs include notations to express temporal sequences and concurrences, as required by a hardware module.

A good example of hardware description language is VHDL, which is an acronym for VHSIC Hardware Description Language. VHSIC stands for Very High Speed Integrated Circuits and it was a government program of USA in the early 80's. Later, the VHDL language became an IEEE (Institute of Electrical and Electronic Engineers) standard and nowadays there are several tools to simulate and synthesize (i.e., generate hardware) circuits described in VHDL. Other hardware description languages are SystemC, Handel-C, Verilog, and System Verilog.

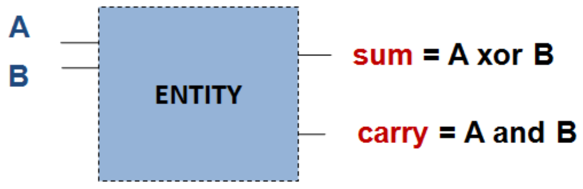
In VHDL, the design of a digital circuit can be described in two abstraction levels, that is, structural or behavioral. Descriptions at the register transfer level (RTL) are widely used to develop digital systems. VHDL is not equivalent to a software programming language and the synthesis tools are not equal to the compiler tools, since they do not generate executable code from a VHDL description. Additionally, these descriptions can be used to generate a piece of hardware, for example, a file for configuring an FPGA.

VHDL descriptions can be simulated, that is, run in a simulator. The testing process can be performed via *testbenches* where stimuli are generated for simulating VHDL descriptions. A testbench defines the external stimulus to be used as test cases to a circuit input. The testbenches can be written in VHDL or in many other languages (e.g. C, C++, and SystemC).

The VHDL models consist of two main parts, the *entity* and the *architecture*. As shown in Figure 2.1, the entity part of a VHDL description has only the input and output pins (interface) to be used by the circuit. It has no information at all regarding the circuit's internal logic (circuit operation). In the architecture part, as shown in Figure 2.2, there is the operational description of the circuit. Figure 2.3 shows the complete VHDL description for the half-adder circuit. It is important to notice that in this example, the *std_logic* type has been used and, for this reason, its library has to be included in the description. As a result, a VHDL description usually has the following sessions:

- **Library** definition - includes the required libraries and packages, for instance, the IEEE package (see lines 1 - 2 in Figure 2.3);
- **Entity** - defines the "pins" of the digital circuit (signals), i.e. the interface between the implemented logic and the external world (see lines 4 - 11 in Figure 2.3);
- **Architecture** - defines the functionality of a digital circuit, using the input and output "pins" that are listed at the entity part (see lines 13 - 17 in Figure 2.3). Thus, an entity may have different implementations (i.e., architectures) for the same function.

ENTITY

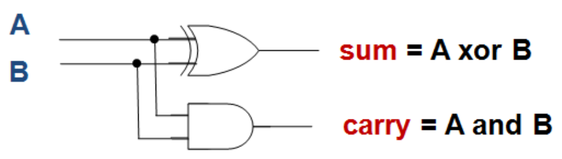


A	B	F1	F2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
entity halfadd is
port (A: in std_logic;
      B: in std_logic;
      sum: out std_logic;
      carry: out std_logic
      );
end halfadd;
```

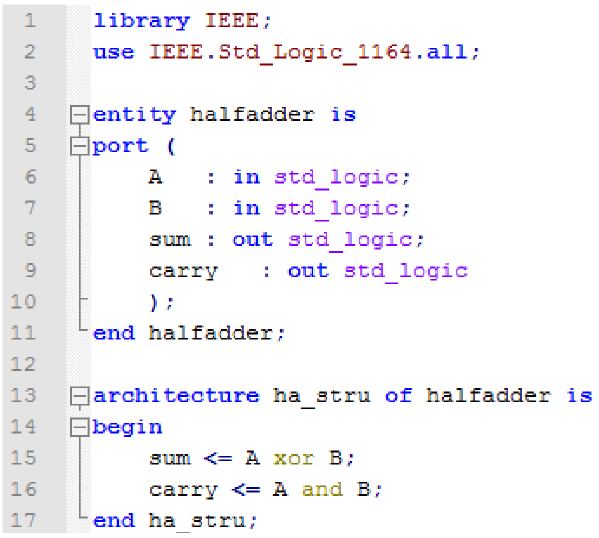
Figure 2.1. "Entity" declaration for the half-adder circuit.

ARCHITECTURE



```
architecture ha_stru of halfadd is
begin
  sum <= A xor B;
  carry <= A and B;
end ha_stru;
```

Figure 2.2. "Architecture" declaration for the half-adder circuit.



```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity halfadder is
port (
     A    : in std_logic;
     B    : in std_logic;
     sum : out std_logic;
     carry    : out std_logic
     );
end halfadder;

architecture ha_stru of halfadder is
begin
     sum <= A xor B;
     carry <= A and B;
end ha_stru;
```

Figure 2.3. Example of entity and architecture in VHDL. *sum* and *carry* are assigned in parallel.

2.2 Laboratory Assignment

The laboratory objectives are:

- to allow the students to have a first contact with the VHDL language;
- to provide a first contact with an EDA tool;
- to continue the training and understanding of a complete FPGA design flow, but this time starting from a VHDL description design entry.

Laboratory Session

The tasks to be completed in this laboratory session, using Altera's Quartus II EDA tool, are as follows:

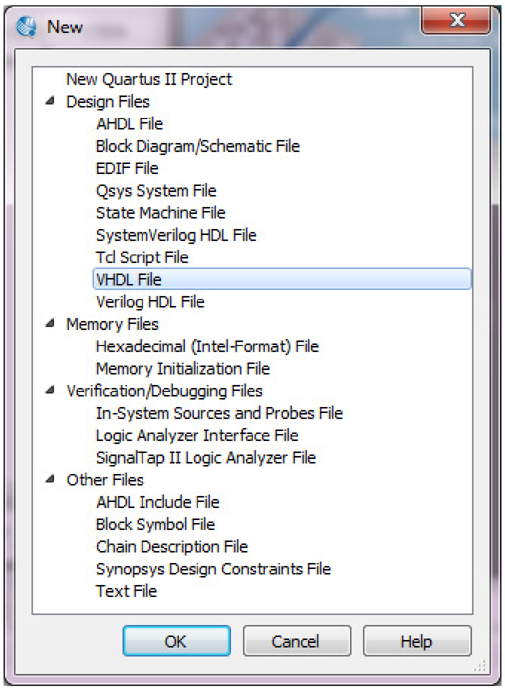
- Create the half-adder circuit shown in Figure 2.3 using the VHDL design entry editor;
- Perform the synthesis;
- Fix simulation and synthesis errors;
- Prototype and test the half-adder circuit in the FPGA board.

Step 1 – Creating a new project

This step is exactly the same as "Step 1" described in Chapter 1. It is important to remember to type "halfadder" as the project's name (page 1 of 5 in the New Project Wizard – see Figure 1.13).

Step 2 – Design entry (VHDL)

Chose *File –> New* and *VHDL File*, as shown in Figure 2.4. This will open an empty text editor.


New

New Quartus II Project
Design Files
AHDL File
Block Diagram/Schematic File
EDIF File
Qsys System File
State Machine File
SystemVerilog HDL File
Tcl Script File
VHDL File
Verilog HDL File
Memory Files
Hexadecimal (Intel-Format) File
Memory Initialization File
Verification/Debugging Files
In-System Sources and Probes File
Logic Analyzer Interface File
SignalTap II Logic Analyzer File
Other Files
AHDL Include File
Block Symbol File
Chain Description File
Synopsys Design Constraints File
Text File

OK Cancel Help

Figure 2.4. New VHDL file.

Type in the text editor the half-adder VHDL code shown in Figure 2.3, and save it: *File –> Save* (or just ctrl-S). The default name to save the file is halfadder.vhd, as it is the name provided as the project name in Step 1. The file will be saved, also in the default location, in the project folder defined in Step 1. Be sure that the check box “Add file to current project” is checked. Again, at this point it is important to be sure that the entity’s name is also “halfadder”, otherwise, the synthesis tool will not be able to find the entity to be synthesized.

Step 3 – Synthesis

This step is the same as “Step 3” described in Chapter 1, that is, choose *Processing –> Start Compilation* (or Ctrl-L). In case of errors, read carefully the “Processing” messages (bottom window), and fix them in the VHDL source code (in the text editor). There will be some warning messages, and at this point look just for the “critical” ones.

Step 4 – Simulation

Before starting the simulation, see “Step 4” in Chapter 1 in order to configure simulator options. After that, to run ModelSim, in Quartus II menu choose *Tools –> Run Simulation Tool –> RTL Simulation* (or press the 7th icon from right to left shown in Figure 1.21). Next, follow the instructions provided in Chapter 1, “Step 4”:

- Simulate –> Start Simulation;
- In the "Start Simulation" window, look for the "Design" tab (see Figure 1.24);
- In the "work" Library, click on the "+" sign, and select the "halfadder" entity as shown in Figure 1.24;
- Click OK to start the simulation;
- Select the *A*, *B*, *sum* and *carry* signals in the Objects window, and add them to the Wave window, as shown in Figure 2.5;
- Set the *A* signal to 0, and the *B* signal to 0, using the "Force" option as explained in Chapter 1, "Step 4". Set the simulation run length to 100 ps, and press the *Run* button. Next, change *A* to 0 and *B* to 1, and run for another 100 ps. Change *A* to 1 and *B* to 0, and run for 100 ps. Finally, change *A* to 1 and *B* to 1, and run for 100 ps.
- The expected simulation results are shown in Figure 2.6.

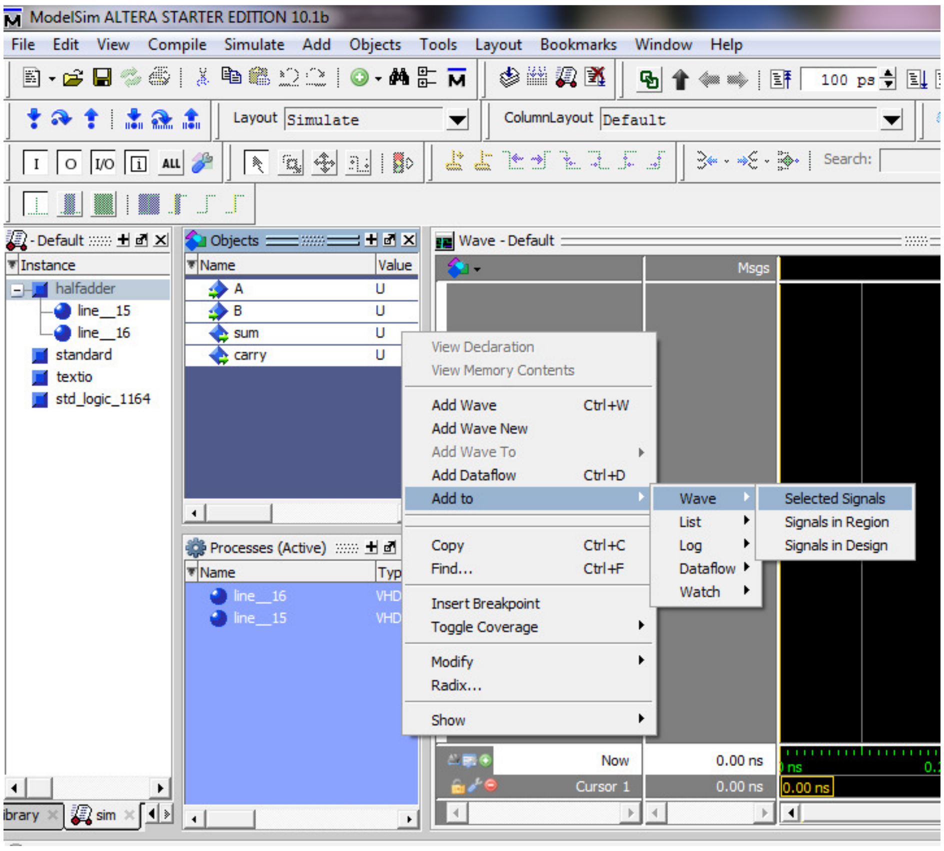


Figure 2.5. Selecting signals from the half-adder design.

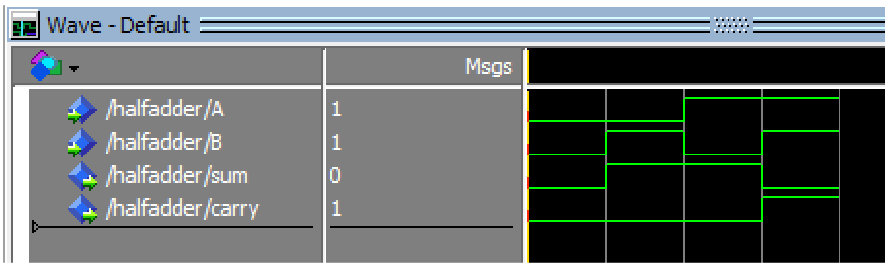


Figure 2.6. Simulation results for the half-adder in VHDL.

Step 5 – FPGA prototyping

The first task in order to build a physical design to prototype in an FPGA board is to specify which entity signal goes to which FPGA pin. For instance, in the half-adder design, if nothing is done regarding the *A*, *B*, *sum* and *carry* signals, the synthesis tool will randomly allocate pins in the FPGA device, and the design may not work in the target board. In Chapter 1 the half-adder entity signals were assigned to the physical (FPGA) pins using the Pin Planner tool (see Figure 1.31). This is an adequate strategy for designs with few external interface pins, which is the case of the half-adder circuit. However, this is not the case in real world designs were dozens or even hundreds of signals are used in their top-level entities. For this reason, for the VHDL version of the half-adder design an automatic pin assignment strategy will be adopted.

The listing of all pins available in the DE2 board is included in a text file. In this listing, each pin has a specific name, and the signals of the half-adder entity should be renamed according to the FPGA board nomenclature. Alternatively, the pin names in the file can be renamed to match the signal names in the design entity, but this is not a common practice.

Each FPGA vendor has its own format, but usually only the relation between the signal name and the target pin is specified. There is no indication regarding the pin direction (input, output).

Xilinx pin assignment files have the ".ucf" extension, and lines such as:

```
NET "sum" LOC = "P28";
NET "carry" LOC = "P110";
```

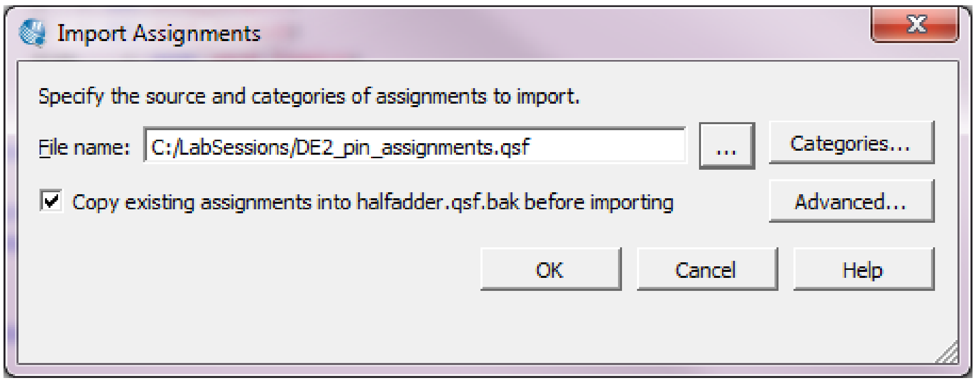
In this case, the sum and carry signals of the half-adder entity will be assigned to FPGA pins 28 and 110, respectively.

Altera has a different format for their pin assignments files, as shown next:

```
set_location_assignment PIN_N25 -to A
set_location_assignment PIN_N26 -to B
set_location_assignment PIN_AE23 -to sum
set_location_assignment PIN_AF23 -to carry
```

In this example, *A* and *B* are assigned in the DE2 board to switch 0 and switch 1, respectively. The signals *sum* and *carry* are assigned to LED red 0 and LED red 1, respectively.

The full list of available pins on the board is included in the *DE2_pin_assignments.qsf* file, which can be found in the official website for this book. The Quartus II Setting File (".qsf") can be downloaded from the book's website and copied to the project's folder (e.g., C:\LabSessions\halfadder), the FPGA pins listed in this file must be imported into the design: menu *Assignments –> Import Assignments*. Browse to find the folder, and press OK as shown in Figure 2.7.


Import Assignments

Specify the source and categories of assignments to import.

File name: C:/LabSessions/DE2_pin_assignments.qsf

[✓] Copy existing assignments into halfadder.qsf.bak before importing

Figure 2.7. Import assignments for the half-adder design.

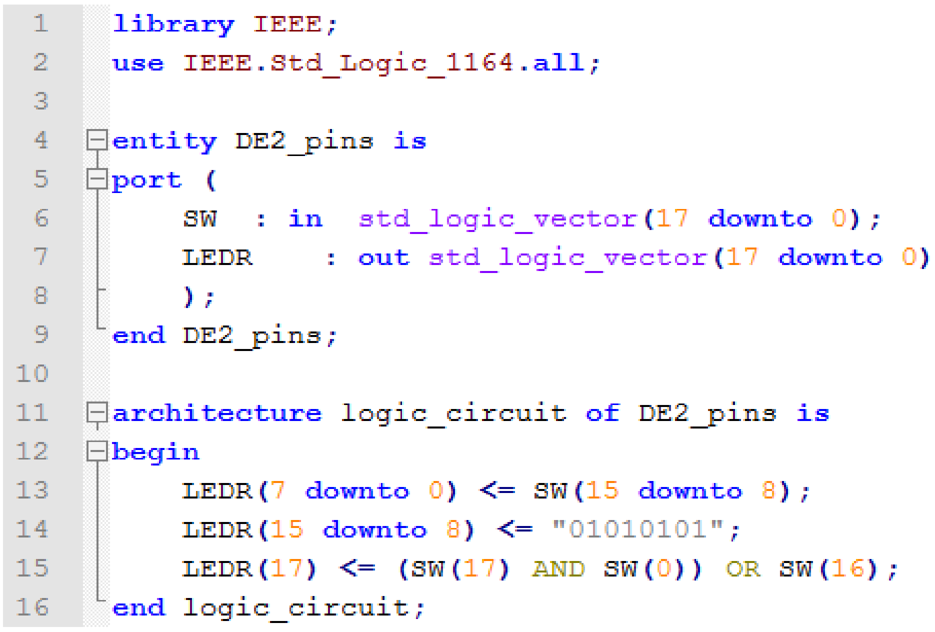
This file has 425 pins listed, and each pin is assigned to a pre-defined label. The N25, N26, AE23 and AF23 FPGA pins are assigned to the labels SW[0], SW[1], LEDR[0] and LEDR[1], as shown next:

```
set_location_assignment PIN_N25 -to SW[0]
set_location_assignment PIN_N26 -to SW[1]
…
set_location_assignment PIN_AE23 -to LEDR[0]
set_location_assignment PIN_AF23 -to LEDR[1]
…
```

In the DE2 board Printed Circuit Board (PCB) design, the FPGA pin N25 is connected to the switch SW_0 shown at the bottom of Figure 1.34. The FPGA pin N26 is connected to switch SW_1, and so on.

In the *DE2_pin_assignments.qsf* file, the label SW represents a vector with 18 positions (0 to 17). LEDR is also a vector with 18 positions.

To perform the connection between the physical world (DE2 switches and LEDs) and internal FPGA signals, a VHDL entity should include the input and output pins available on the board. As shown in Figure 2.8, on line 6 the label SW is used. SW is an 18 bits vector, and it has the same name (and size) as the SW symbol connected to the FPGA switch pins listed in the *DE2_pin_assignments.qsf* file. On line 7, label LEDR is used in order to match the name used in the pin assignments file. The VHDL commands in lines 13 to 15 write to the red LEDs: switch values (line 13); a constant value (line 14), and the results of a logical operation (line 15).



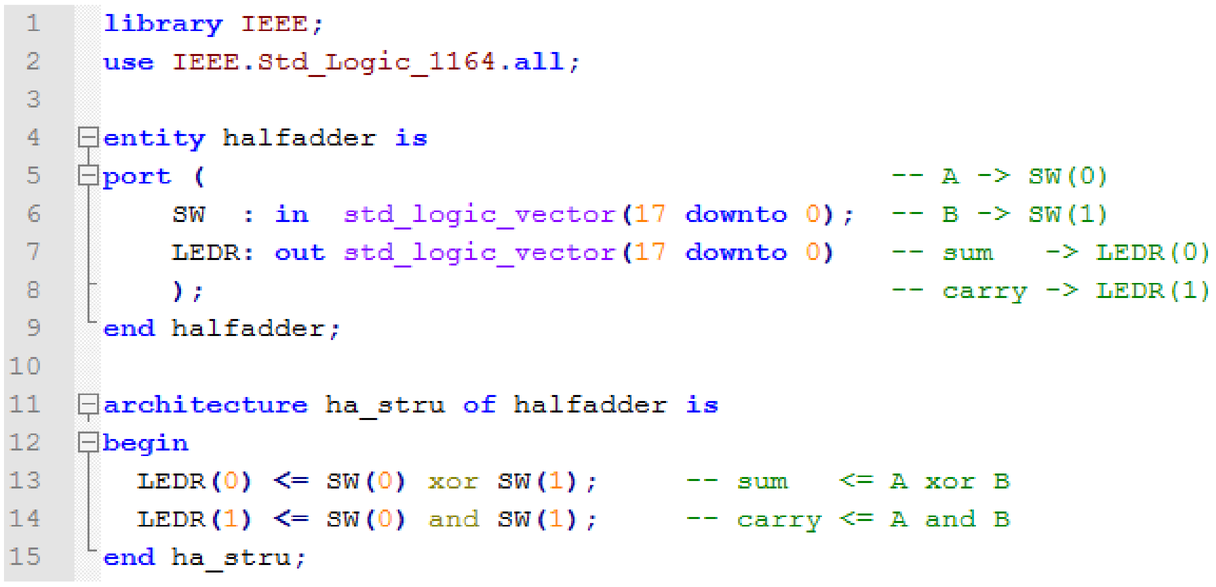
```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity DE2_pins is
port (
    SW   : in  std_logic_vector(17 downto 0);
    LEDR     : out std_logic_vector(17 downto 0)
    );
end DE2_pins;

architecture logic_circuit of DE2_pins is
begin
    LEDR(7 downto 0) <= SW(15 downto 8);
    LEDR(15 downto 8) <= "01010101";
    LEDR(17) <= (SW(17) AND SW(0)) OR SW(16);
end logic_circuit;
```

Figure 2.8. DE2 pins declared in an entity.

In order to adapt the half-adder circuit to the DE2 board, the design's entity should be changed according to the *DE2_pin_assignments.qsf* file contents. The changes consist in renaming the *A*, *B*, *sum* and *carry* signals to *SW(0)*, *SW(1)*, *LEDR(0)*, and *LEDR(1)*, respectively. The changes are shown in Figure 2.9.



```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity halfadder is
port (                                             -- A -> SW(0)
    SW   : in  std_logic_vector(17 downto 0);     -- B -> SW(1)
    LEDR: out std_logic_vector(17 downto 0)       -- sum   -> LEDR(0)
    );                                             -- carry -> LEDR(1)
end halfadder;

architecture ha_stru of halfadder is
begin
  LEDR(0) <= SW(0) xor SW(1);      -- sum   <= A xor B
  LEDR(1) <= SW(0) and SW(1);      -- carry <= A and B
end ha_stru;
```

Figure 2.9. Half-adder design adapted to DE2 board pin names.

After have concluded the modifications in the VHDL source code, perform a new synthesis (Compile button in Quartus II menu), and download the generated configuration file to the FPGA. The steps to perform the download are the same as "Step 5" in Chapter 1 (see Figure 1.32 and Figure 1.33).

To test the circuit, follow the instructions provided at the end of Chapter 1, using the switches and LEDs as shown in Figure 1.34, where *A* is *SW(0)*, *B* is *SW(1)*, *sum* is *LEDR(0)*, and *carry* is *LEDR(1)*.

Going Beyond

The DE2 board has an LCD which can be programmed and written manually, using the available switches. The VHDL code to access the LCD pins is shown in Figure 2.10.

```
library ieee;
use ieee.std_logic_1164.all;
entity LCD is
port (
      LCD_DATA    : out std_logic_vector(7 downto 0);
      LCD_RW      : out std_logic;
      LCD_EN      : out std_logic;
      LCD_RS      : out std_logic;
      LCD_ON      : out std_logic;
      LCD_BLON    : out std_logic;
      SW          : in std_logic_vector(17 downto 0)
);
end LCD;
architecture LCD_WR of LCD is
begin
      LCD_ON    <= SW(17);
      LCD_BLON  <= SW(16);
      LCD_DATA  <= SW(7 downto 0);
      LCD_RS    <= SW(8);
      LCD_EN    <= SW(9);
      LCD_RW    <= SW(10);
end LCD_WR;
```

Figure 2.10. VHDL code for LCD writing in DE2 board.

After have synthesized and downloaded the generated configuration file to the FPGA board, follow the next steps to write control and data to the LCD.

Step 1 – write command 1 (38H) – switches the LCD on, cursor on, and blink on

Switch (SW)	Value (switch position)	Effect
17	1	LCD_ON
16	1	LCD_BLON
7..0	0011 1000	Command
8	0	LCD_RS (0 = control)
9	0 → 1 → 0	LCD_EN
10	0	LCD_RW

Step 2 – write command 2 (0FH) – switches the LCD on, cursor on, and blink on

Switch (SW)	Value (switch position)	Effect
17	1	LCD_ON
16	1	LCD_BLON
7..0	0000 1111	Command
8	0	LCD_RS (0 = control)
9	$0 \rightarrow 1 \rightarrow 0$	LCD_EN
10	0	LCD_RW

Step 3 – write command 3 (06H) – switches the LCD on, cursor on, and blink on

Switch (SW)	Value (switch position)	Effect
17	1	LCD_ON
16	1	LCD_BLON
7..0	0000 0110	Command
8	0	LCD_RS (0 = control)
9	$0 \rightarrow 1 \rightarrow 0$	LCD_EN
10	0	LCD_RW

Step 4 – write data to LCD

Switch (SW)	Value (switch position)	Effect
17	1	LCD_ON
16	1	LCD_BLON
7..0	0100 0001	'A' or 41H (ASCII)
8	1	LCD_RS (1 = data)
9	$0 \rightarrow 1 \rightarrow 0$	LCD_EN
10	0	LCD_RW

Step 5 – clear the LCD

Switch (SW)	Value (switch position)	Effect
17	1	LCD_ON
16	1	LCD_BLON
7..0	0000 0001	Clear command
8	0	LCD_RS (0 = control)
9	$0 \rightarrow 1 \rightarrow 0$	LCD_EN
10	0	LCD_RW