Using the *DESim* Application with Verilog Code

This tutorial introduces the *DESim* application, which you can use to simulate circuits specified with Verilog code. The *DESim* application provides a *graphical user interface* (GUI) that represents some of the features of a *DE1-SoC* board. This GUI serves as a “front end” for the *ModelSim* simulator. Using the *DESim* GUI you can invoke both the ModelSim Verilog *compiler* and *simulator*. Inputs to the *ModelSim* simulator can be provided by clicking on features in the *DESim* GUI, which also shows results produced by the simulator on displays that look like the ones on a DE1-SoC board.

Contents:

Requirements:

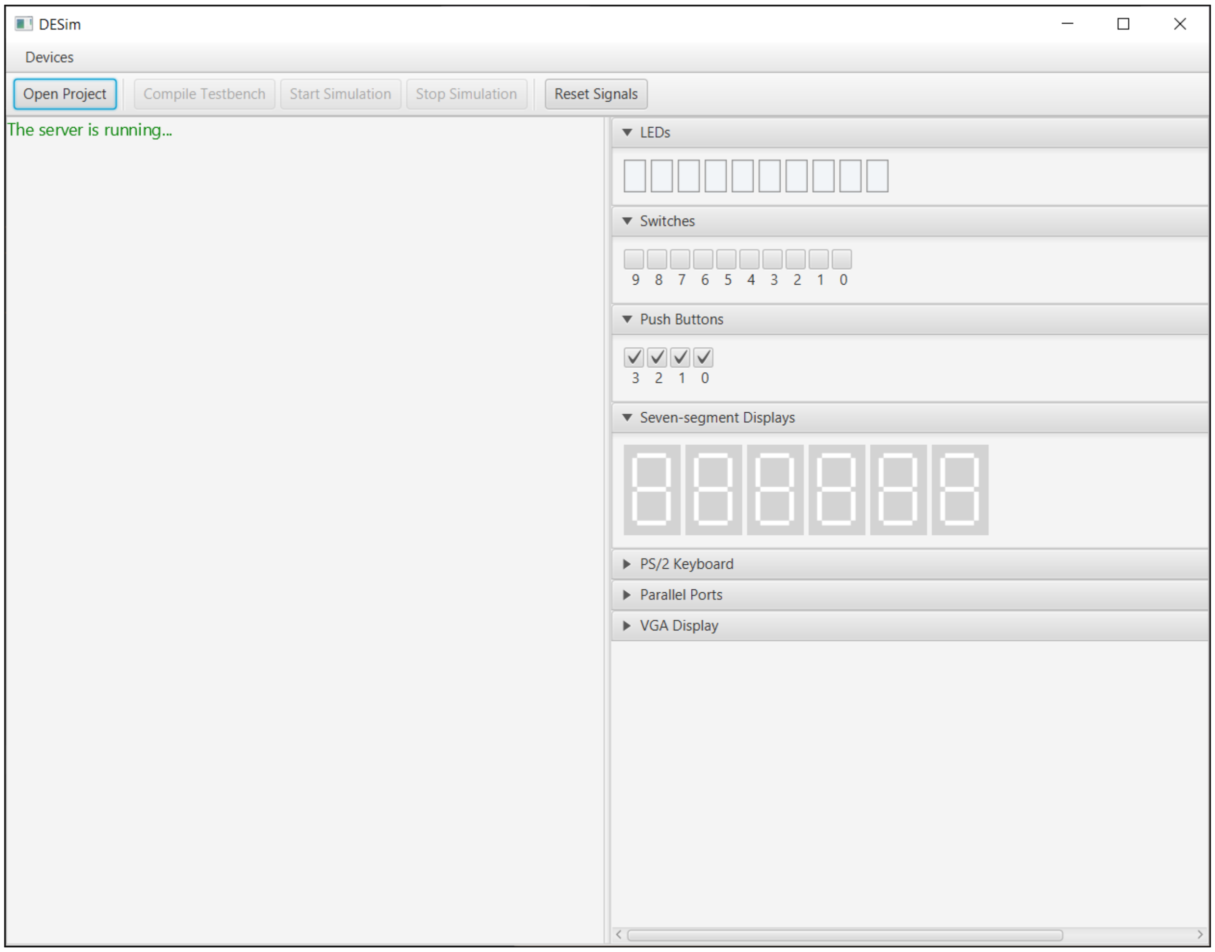
- A computer running Microsoft® Windows® (version 10 is recommended).
- A good working knowledge of Verilog code.
- ModelSim-Intel FPGA Starter Edition software, version 10.5b. This software must be installed on the computer that is running the *DESim* software. The required *ModelSim* software is part of the *Quartus Prime* suite of CAD tools provided by Intel® Corporation. Version 10.5b of ModelSim is packaged with a number of *Quartus* software releases, including 18.0, 18.1, and 19.0.
- You should know how to use *ModelSim*® to simulate Verilog code using a *testbench*. This material is presented in the tutorial *Using the ModelSim-Intel FPGA Simulator with Verilog Testbenches*.
- The *DESim* application. Instructions for downloading and installing the *DESim* application are available in the document *Installing the DESim Application*, available from the Intel FPGA Academic Program.

Getting Started

Start the *DESim* program to reach the graphical user interface (GUI) shown in Figure 1. You should see the message “The server is running...” at the top of the *message pane* in the GUI. If you do not see this message, but instead you see a message Server setup failed, then the *DESim* software is not working properly and should be closed. In this case see the troubleshooting section at the end of this document.

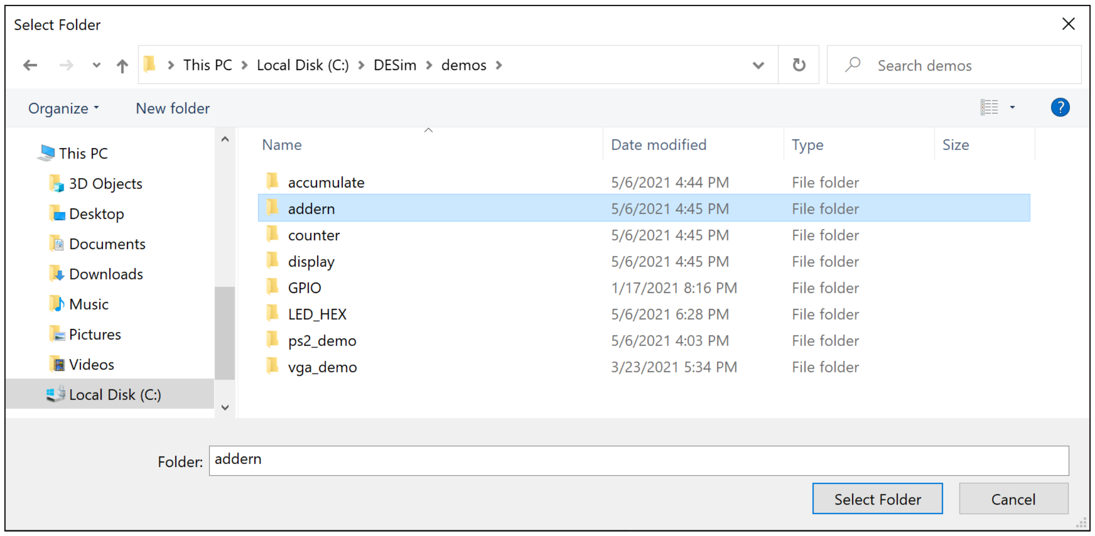
On the right-hand side of Figure 1 the `LEDs` represent the red lights `LEDR`$_{9-0}$ that are provided on a DE1-SoC board. The `Switches` correspond to the board’s `SW`$_{9-0}$ slide switches, the `Push Buttons` to `KEY`$_{3-0}$, and the `Seven-segment Displays` to `HEX5`, `HEX4`, $\cdots$, `HEX0`. There are also some additional features in the GUI called `PS/2 Keyboard`, `Parallel Ports`, and `VGA Display`. These features of the *DESim* GUI are not described in this document.

The *DESim* tool works in the context of a *project*. To introduce the features of the *DESim* GUI we will first open an existing project. This example is a multibit adder named *addern*, which is provided as a *demo* project along with the *DESim* software. Referring to Figure 1, click on the `Open Project` command to reach the dialogue displayed in Figure 2. As shown in the figure, navigate to the `demos` folder, click to select the `addern` project, and then click the `Select Folder` button.



The server is running...

Figure 1: The *DESim* GUI.

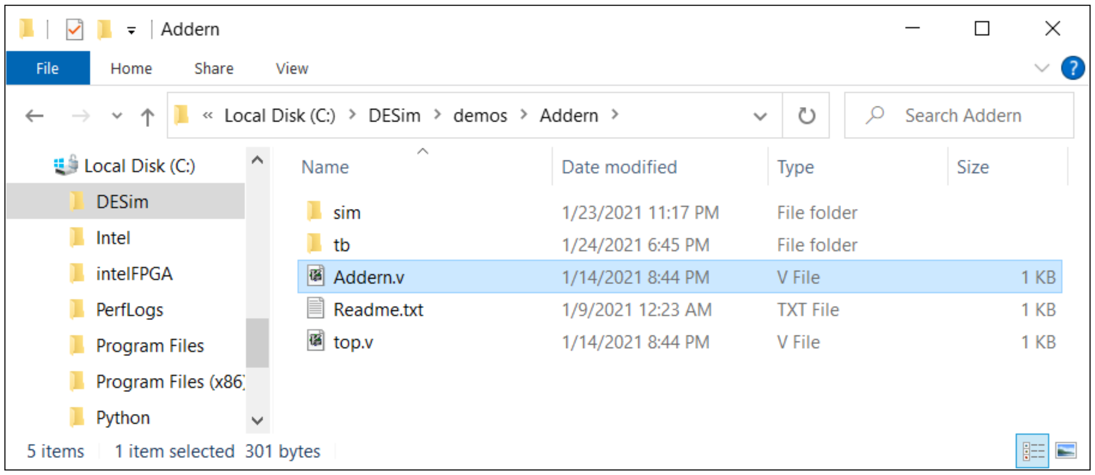


Name	Date modified	Type	Size
accumulate	5/6/2021 4:44 PM	File folder	
addern	5/6/2021 4:45 PM	File folder	
counter	5/6/2021 4:45 PM	File folder	
display	5/6/2021 4:45 PM	File folder	
GPIO	1/17/2021 8:16 PM	File folder	
LED_HEX	5/6/2021 6:28 PM	File folder	
ps2_demo	5/6/2021 4:03 PM	File folder	
vga_demo	3/23/2021 5:34 PM	File folder	

Folder: addern

Figure 2: Opening the *addern* project.

Using Microsoft Windows `File Explorer`, Figure 3 shows the contents of the file-system folder that holds the *addern* project. It consists of the folders named `sim` and `tb`, as well as the files called *Addern.v*, and *top.v*. There is also a *Readme.txt* file, but it just provides documentation and is not really a part of the *DESim* project.



Name	Date modified	Type	Size
sim	1/23/2021 11:17 PM	File folder	
tb	1/24/2021 6:45 PM	File folder	
Addern.v	1/14/2021 8:44 PM	V File	1 KB
Readme.txt	1/9/2021 12:23 AM	TXT File	1 KB
top.v	1/14/2021 8:44 PM	V File	1 KB

Figure 3: The *addern* folder.

The *Addern.v* file, shown in Figure 4, is the Verilog code that will be simulated in this part of the tutorial. We will use the *DESim* GUI to specify signal values for the adder's inputs, *Cin*, *X*, and *Y*, and then display the simulation results produced for the outputs, *Sum* and *Cout*, on the `LED`s. To make connections between the adder's ports and the signals that are associated with the *DESim* GUI we *instantiate* the *Addern* module in another Verilog module called *top*. This module is defined in the file *top.v*, displayed in Figure 5. Its ports use the signal names that are appropriate for a top-level Verilog module which is intended to be implemented on a DE1-SoC board. These port names include `CLOCK_50`, `SW`, `KEY`, `HEX0`, $\cdots$, `HEX5`, and `LEDR`. For the *Addern* module we only use some of these ports, and leave the others unconnected (the unused ports are included for consistency with other *DESim demo* projects).

The *Addern* module is instantiated in Line 15 in *top.v* by the statement

```
Addern U1 (SW[9], SW[3:0], SW[7:4], LEDR[3:0], LEDR[4]);
```

This statement connects the switch SW_9 to the multibit adder's carry-in, *Cin*, and it connects SW_{3-0} and SW_{7-4} to the adder's *X* and *Y* data inputs, respectively. The *Sum* output is attached to $LEDR_{3-0}$, and the carry-out, *Cout*, is connected to $LEDR_4$.

To compile the *addern* project, in the *DESim* GUI click the `Compile Testbench` command. This command executes a *batch* file called *run_compile.bat*, which is found in the `sim` folder of the *addern* project. This batch file comprises some *ModelSim* commands, shown below:

```
if exist work rmdir /S /Q work

vlib work
vlog ../tb/*.v
vlog ../*.v
```

The batch file executes the *vlib* command, which is part of the *ModelSim* software, to create a `work` folder (first deleting this folder if it already exists). The batch file then invokes the *ModelSim* Verilog compiler, *vlog*,

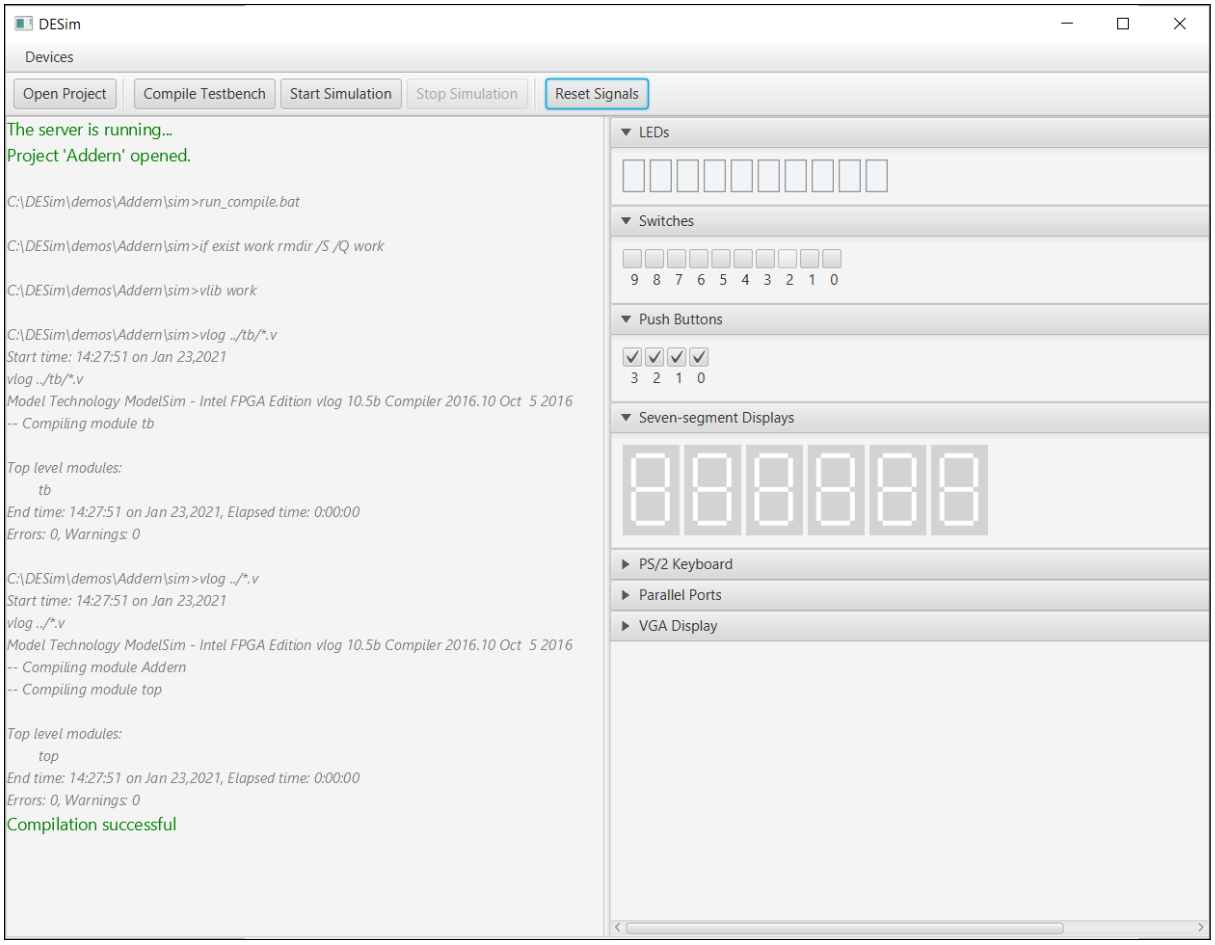
```
// A multi-bit adder
module Addern (Cin, X, Y, Sum, Cout);
    parameter n = 4;
    input Cin;
    input [n-1:0] X, Y;
    output [n-1:0] Sum;
    output Cout;

    assign {Cout, Sum} = X + Y + Cin;
endmodule
```

Figure 4: Verilog code for the multibit adder.

```
module top (CLOCK_50, SW, KEY, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);

    input CLOCK_50;               // DE-series 50 MHz clock signal
    input wire [9:0] SW;          // DE-series switches
    input wire [3:0] KEY;         // DE-series pushbuttons
    output wire [9:0] LEDR;       // DE-series LEDs

    output wire [6:0] HEX0;       // DE-series HEX displays
    output wire [6:0] HEX1;
    output wire [6:0] HEX2;
    output wire [6:0] HEX3;
    output wire [6:0] HEX4;
    output wire [6:0] HEX5;

    Addern U1 (SW[9], SW[3:0], SW[7:4], LEDR[3:0], LEDR[4]);

endmodule
```

Figure 5: Verilog code for the *top* module.

twice. The first invocation of the compiler, `vlog ../tb/*.v`, compiles the Verilog source code in the *addern* project's `tb` folder. This folder holds the *testbench* for the project, which is described shortly. The second call to *vlog* compiles the source-code of the *DESim* project, which includes the files *Addern.v* and *top.v*. Any messages produced while executing *run_compile.bat* are displayed inside the message pane in the *DESim* GUI, as illustrated in Figure 6.



```
The server is running...
Project 'Addern' opened.

C:\DESim\demos\Addern\sim>run_compile.bat

C:\DESim\demos\Addern\sim>if exist work rmdir /S /Q work

C:\DESim\demos\Addern\sim>vlib work

C:\DESim\demos\Addern\sim>vlog ../tb/*.v
Start time: 14:27:51 on Jan 23,2021
vlog ../tb/*.v
Model Technology ModelSim - Intel FPGA Edition vlog 10.5b Compiler 2016.10 Oct  5 2016
-- Compiling module tb

Top level modules:
    tb
End time: 14:27:51 on Jan 23,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0

C:\DESim\demos\Addern\sim>vlog ../*.v
Start time: 14:27:51 on Jan 23,2021
vlog ../*.v
Model Technology ModelSim - Intel FPGA Edition vlog 10.5b Compiler 2016.10 Oct  5 2016
-- Compiling module Addern
-- Compiling module top

Top level modules:
    top
End time: 14:27:51 on Jan 23,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
Compilation successful
```

Figure 6: Messages produced by executing *run_compile.bat*.

The testbench file for the *addern* project that is compiled by the command `vlog ../tb/*.v` is called *tb.v*, and is displayed in Figure 7. It is not necessary to modify (or even examine) much of this code to use the *DESim* software, but we describe some of the code here for completeness. The testbench code in the figure has a general structure that allows it to be used to simulate different examples of Verilog code that might be used in various *DESim* projects. Hence, not all of the code in the testbench is needed for the *addern* project. Line 6 declares the testbench module, which is named *tb*. The next several lines in the code declare some signals that are used in the testbench. The statement

```
initial $sim_fpga(CLOCK_50, SW, KEY, LEDR, HEX, key_action, scan_code,
                  ps2_lock_control, VGA_X, VGA_Y, VGA_COLOR, plot, GPIO);
```

is unique to the *DESim* program. It makes use of a special feature of the *ModelSim* software that allows communication with a *custom software function*. In this case the custom function is part of the *DESim* software and is called *sim_fpga*. This function is stored in a file named *simfpga.vpi*, which has to be included in the `sim` folder of each *DESim* project. The *DESim* GUI sends/receives signal values to/from the ModelSim software via the *sim_fpga* function. This *ModelSim* capability is known as the *Verilog Procedural Interface* (VPI).

```
1  `timescale 1ns / 1ns
2  `default_nettype none
3
4  // This testbench is designed to hide the details of using the VPI code
5
6  module tb();
7
8      reg CLOCK_50 = 0;               // DE-series 50 MHz clock
9      reg [9:0] SW = 0;               // DE-series SW switches
10     reg [3:0] KEY = 0;              // DE-series pushbutton keys
11     wire [(8*6) -1:0] HEX;          // HEX displays (six ports)
12     wire [9:0] LEDR;                // DE-series LEDs
13
14     reg key_action = 0;
15     reg [7:0] scan_code = 0;
16     wire [2:0] ps2_lock_control;
17
18     wire [7:0] VGA_X;               // "VGA" column
19     wire [6:0] VGA_Y;               // "VGA" row
20     wire [2:0] VGA_COLOR;           // "VGA pixel" colour (0-7)
21     wire plot;                      // "Pixel" is drawn when this is pulsed
22     wire [31:0] GPIO;               // DE-series GPIO port
23
24     initial $sim_fpga(CLOCK_50, SW, KEY, LEDR, HEX, key_action, scan_code,
25                       ps2_lock_control, VGA_X, VGA_Y, VGA_COLOR, plot, GPIO);
26
27     wire [6:0] HEX0;                // DE-series HEX0 port
28     wire [6:0] HEX1;                // DE-series HEX1 port
29     wire [6:0] HEX2;                // ...
30     wire [6:0] HEX3;
31     wire [6:0] HEX4;
32     wire [6:0] HEX5;
33
34     // create the 50 MHz clock signal
35     always #10
36         CLOCK_50 <= ~CLOCK_50;
37
38     // connect the single HEX port on "sim_fpga" to the six DE-series HEX ports
39     assign HEX[47:40] = {1'b0, HEX0};
40     assign HEX[39:32] = {1'b0, HEX1};
41     assign HEX[31:24] = {1'b0, HEX2};
42     assign HEX[23:16] = {1'b0, HEX3};
43     assign HEX[15: 8] = {1'b0, HEX4};
44     assign HEX[ 7: 0] = {1'b0, HEX5};
45
46     top DUT (.CLOCK_50(CLOCK_50), .SW(SW), .LEDR(LEDR), .KEY(KEY), .HEX0(HEX0),
47              .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5));
48
49 endmodule
```

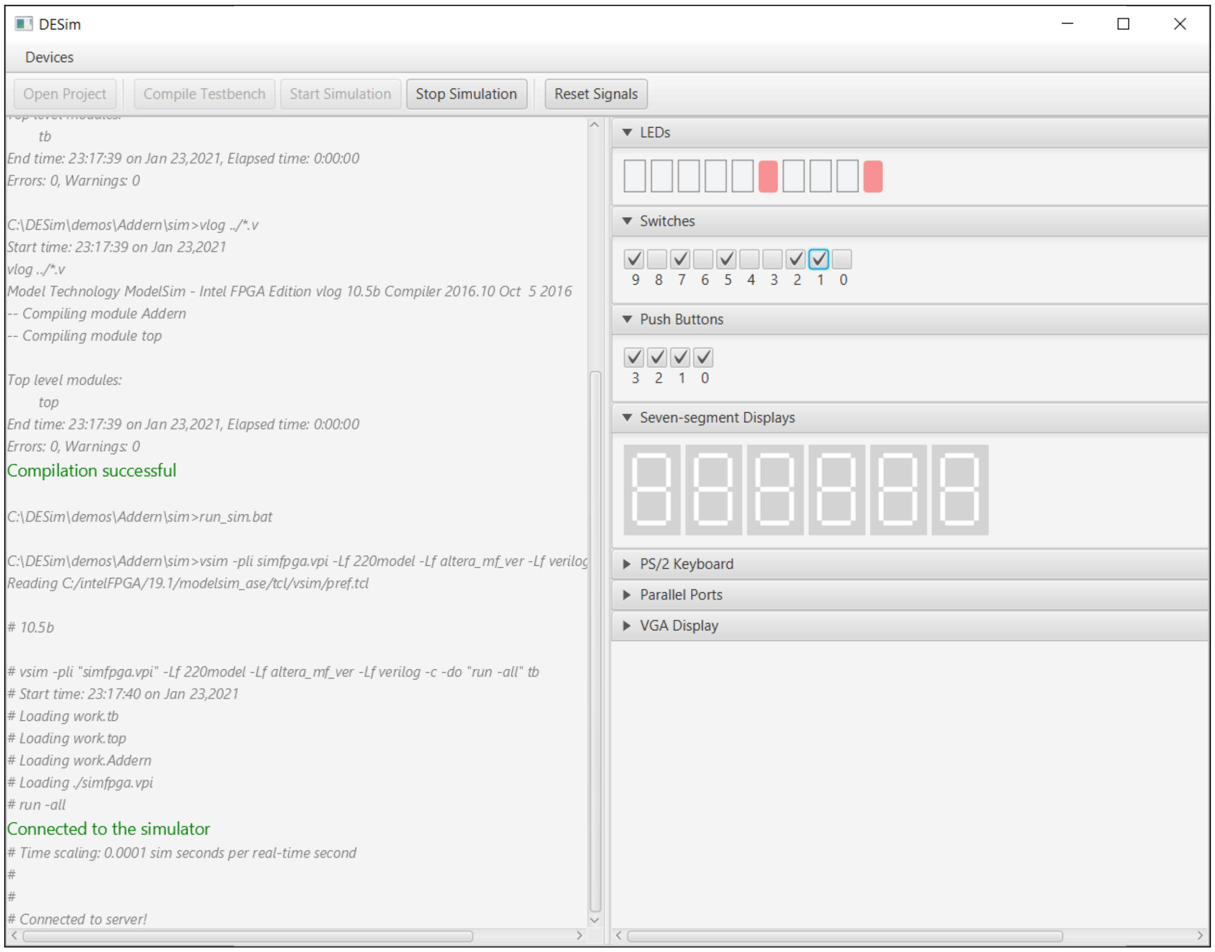
Figure 7: The testbench file, *tb.v*, for the *addern* project.

Line 46 in the testbench code instantiates the *design under test* (DUT), which is the Verilog module named *top* shown in Figure 5. To execute the testbench using the *ModelSim* simulator, click on the `Start Simulation` command in the *DESim* GUI. This command executes a *batch* file called *run_sim.bat*, which is found in the `sim` folder of the *addern* project. This batch file runs *vsim*, the *ModelSim* Verilog simulator, using the command:

```
vsim -pli simfpga.vpi -Lf 220model -Lf altera_mf_ver -Lf verilog -c -do "run -all" tb
```

The `-pli` argument for the *vsim* program instructs it to link to the *sim_fpga* software function that has been (previously) compiled into the *simpfga.vpi* file. The `-L` arguments include some simulation libraries for Intel FPGAs that may be needed by the simulator. Finally, the remaining arguments run the simulation for the top-level module, which is *tb*. Any messages produced while executing *run_sim.bat* are displayed inside the message pane in the *DESim* GUI, as depicted in Figure 8.

As mentioned previously, the *addern* project includes a *Readme.txt* file that documents its usage. This file is displayed in Figure 9. You can follow its instructions to see how the switches and lights are used for the project (of course, you can also find this information by looking at the Verilog source code). An example simulation result is illustrated in Figure 8. It corresponds to $Cin = 1$, $X = (0110)_2 = (6)_{10}$, and $Y = (1010)_2 = (10)_{10}$. The result of the addition is $(10001)_2 = (17)_{10}$ ($Cout = 1$, with $Sum = (0001)_2$), which is displayed on the `LEDs`. Try different settings for the `SW` switches and observe the results displayed on the `LEDs`.



```
      tb
End time: 23:17:39 on Jan 23,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0

C:\DESim\demos\Addern\sim>vlog ../*.v
Start time: 23:17:39 on Jan 23,2021
vlog ../*.v
Model Technology ModelSim - Intel FPGA Edition vlog 10.5b Compiler 2016.10 Oct  5 2016
-- Compiling module Addern
-- Compiling module top

Top level modules:
      top
End time: 23:17:39 on Jan 23,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
Compilation successful

C:\DESim\demos\Addern\sim>run_sim.bat

C:\DESim\demos\Addern\sim>vsim -pli simfpga.vpi -Lf 220model -Lf altera_mf_ver -Lf verilog
Reading C:/intelFPGA/19.1/modelsim_ase/tcl/vsim/pref.tcl

# 10.5b

# vsim -pli "simfpga.vpi" -Lf 220model -Lf altera_mf_ver -Lf verilog -c -do "run -all" tb
# Start time: 23:17:40 on Jan 23,2021
# Loading work.tb
# Loading work.top
# Loading work.Addern
# Loading ./simfpga.vpi
# run -all
Connected to the simulator
# Time scaling: 0.0001 sim seconds per real-time second
#
#
# Connected to server!
```

Figure 8: Messages produced by executing *run_sim.bat*.

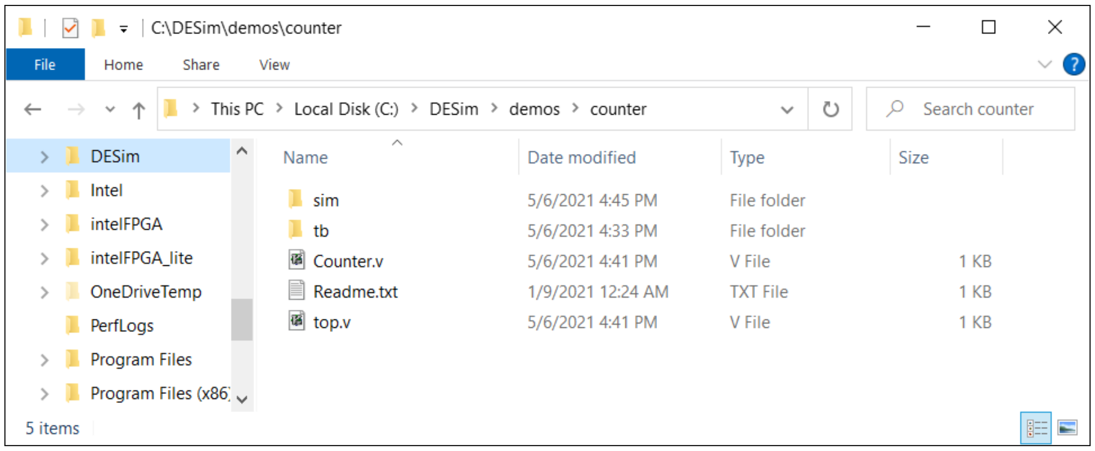
We have now finished discussing the *addern* sample project.

```
To use this demo:

-- set a value, X, using SW[3:0]
-- set a value, Y, using SW[7:4]
-- set a carry-in, Cin, using SW[9]

The circuit produces the 5-bit Sum = X + Y + Cin,
which is displayed on LED[4:0]
```

Figure 9: The *Readme.txt* file for the *addern* project.

Simulating a Sequential Circuit

Another *DESim* sample project called *counter* is included in the *DESim* `demos` folder. Use the `Open Project` command in the *DESim* GUI to open this project. As illustrated in Figure 10 using `File Explorer`, the contents of the file-system folder for this project look the same as for the *addern* project (Figure 3), except that there is a Verilog source-code file named *Counter.v*. Figure 11 shows the contents of the *Counter.v* file. It represents a 24-bit counter with synchronous reset. The port names for this module correspond to the signal names on the DE1-SoC board, with KEY_0 being used for reset, *CLOCK_50* for the clock signal, and *LEDR* for the outputs. Since *LEDR* is a 10-bit signal and the counter has 24 bits, only a subset of the counter outputs (the most-significant ones) are connected to *LEDR*.



Name	Date modified	Type	Size
sim	5/6/2021 4:45 PM	File folder	
tb	5/6/2021 4:33 PM	File folder	
Counter.v	5/6/2021 4:41 PM	V File	1 KB
Readme.txt	1/9/2021 12:24 AM	TXT File	1 KB
top.v	5/6/2021 4:41 PM	V File	1 KB

Figure 10: The *counter* folder.

As described for the *addern* project, the *Counter* module is *instantiated* in another Verilog module called *top*. This module is displayed in Figure 12. It is the same as the one from Figure 5, except that Line 15 instantiates the *Counter* module. The port `KEY[0]` in the `top` module is connected to the counter's reset input, `CLOCK_50` to its clock input, and `LEDR` to the counter's output port.

To compile the *counter* project, in the *DESim* GUI click the `Compile Testbench` command. This command executes the *batch* file called *run_compile.bat*, which is found in the `sim` folder of the *counter* project. This batch file is identical to the one described earlier for the *addern* project. The testbench file that is compiled by *run_compile.bat* for the *counter* project is found in its `tb` folder. This testbench, *tb.v*, is identical to the one shown in Figure 7.

```
module Counter (KEY, CLOCK_50, LEDR);
    input [0:0] KEY;
    input CLOCK_50;
    output [9:0] LEDR;
    parameter n = 24;

    reg [n-1:0] Count;
    wire Clock, Resetn;

    assign Clock = CLOCK_50;
    assign Resetn = KEY[0];

    // the counter
    always @(posedge Clock)
        if (Resetn  == 1'b0)        // synchronous clear
            Count <= 0;
        else
            Count <= Count + 1;

    assign LEDR = Count[n-1:n-10];
endmodule
```

Figure 11: Verilog code for the 24-bit counter.

To execute the testbench for the *counter* project, click on the `Start Simulation` command in the *DESim* GUI. This command executes the *batch* file called *run_sim.bat*. It is identical the one described for the *addern* project and runs the *vsim* Verilog simulator.

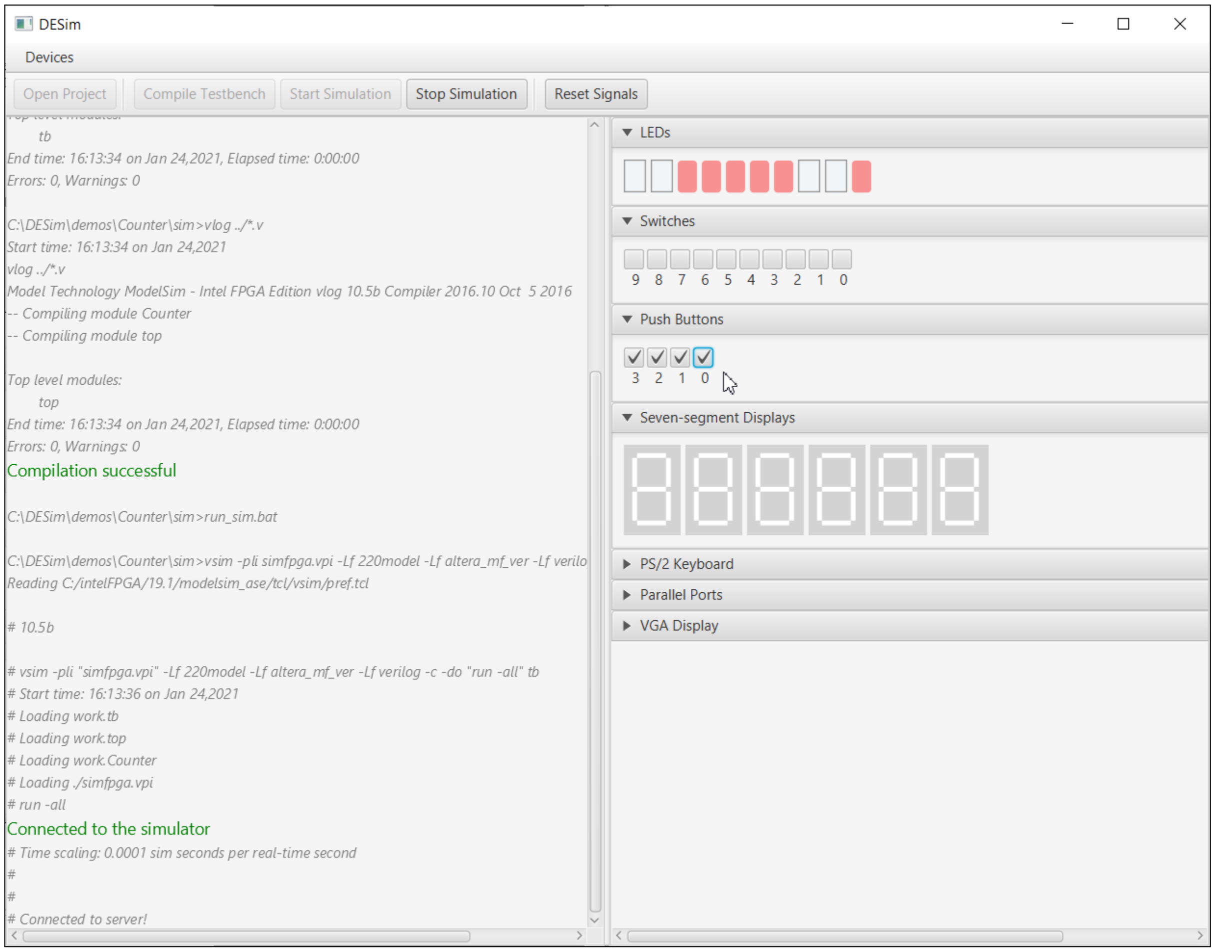
```
module top (CLOCK_50, SW, KEY, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);

    input CLOCK_50;             // DE-series 50 MHz clock signal
    input wire [9:0] SW;        // DE-series switches
    input wire [3:0] KEY;       // DE-series pushbuttons
    output wire [9:0] LEDR;     // DE-series LEDs

    output wire [6:0] HEX0;     // DE-series HEX displays
    output wire [6:0] HEX1;
    output wire [6:0] HEX2;
    output wire [6:0] HEX3;
    output wire [6:0] HEX4;
    output wire [6:0] HEX5;

    Counter U1 (KEY[0], CLOCK_50, LEDR);

endmodule
```

Figure 12: The *top* module for the *counter* project.

The *Readme.txt* file for the *counter* project specifies:

```
To use this demo, reset the circuit by pressing and releasing KEY[0].
```

In the *DESim* GUI when the `Push Buttons` have a check mark shown, they are set to the value 1. To reset the counter circuit, click $\texttt{KEY}_0$ once to *press* this button (this action sets the corresponding signal for this button to 0), and then click it again to *release* the button. The 24-bit counter will start to operate and the ten most-significant counter outputs will be displayed on the `LEDs`. A screen-shot of the *DESim* GUI while simulating the *counter* project is shown in Figure 13.

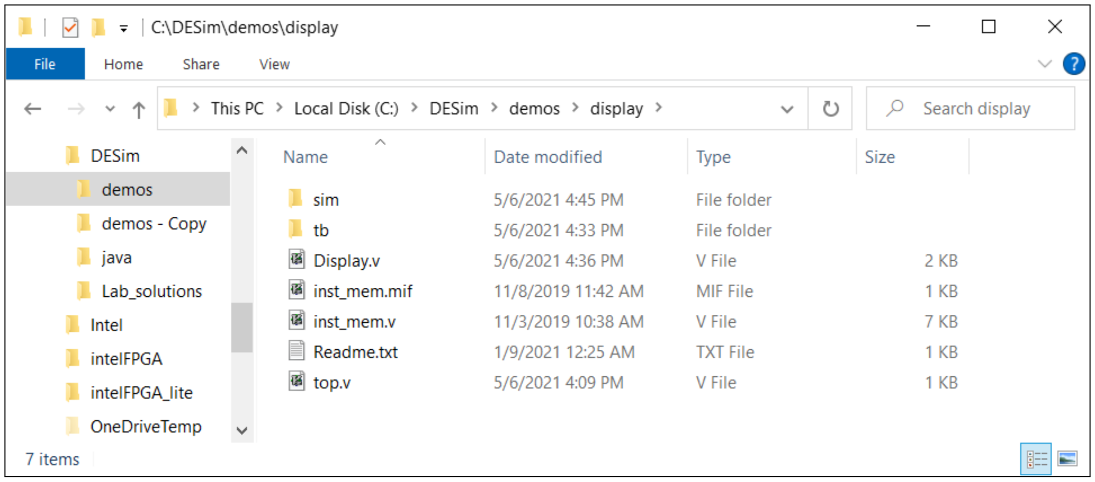


```
      tb
End time: 16:13:34 on Jan 24,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0

C:\DESim\demos\Counter\sim>vlog ../*.v
Start time: 16:13:34 on Jan 24,2021
vlog ../*.v
Model Technology ModelSim - Intel FPGA Edition vlog 10.5b Compiler 2016.10 Oct  5 2016
-- Compiling module Counter
-- Compiling module top

Top level modules:
      top
End time: 16:13:34 on Jan 24,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
Compilation successful

C:\DESim\demos\Counter\sim>run_sim.bat

C:\DESim\demos\Counter\sim>vsim -pli simfpga.vpi -Lf 220model -Lf altera_mf_ver -Lf verilo
Reading C:/intelFPGA/19.1/modelsim_ase/tcl/vsim/pref.tcl

# 10.5b

# vsim -pli "simfpga.vpi" -Lf 220model -Lf altera_mf_ver -Lf verilog -c -do "run -all" tb
# Start time: 16:13:36 on Jan 24,2021
# Loading work.tb
# Loading work.top
# Loading work.Counter
# Loading ./simfpga.vpi
# run -all
Connected to the simulator
# Time scaling: 0.0001 sim seconds per real-time second
#
#
# Connected to server!
```

Figure 13: Simulating the *counter* project.

Simulating a Circuit that Includes a Memory Module

The *DESim* `demos` folder includes a project called *display*. It shows how to instantiate a memory module in Verilog code, and how to initialize the stored contents of the memory in a *DESim* simulation. Use the `Open Project` command to open this example project. As illustrated in Figure 14, the contents of the file-system folder for this project look similar to the previous ones, but there are two extra files: *inst_mem.v* and *inst_mem.mif*. These files are used for the memory module in the circuit, which is described shortly.



Name	Date modified	Type	Size
sim	5/6/2021 4:45 PM	File folder	
tb	5/6/2021 4:33 PM	File folder	
Display.v	5/6/2021 4:36 PM	V File	2 KB
inst_mem.mif	11/8/2019 11:42 AM	MIF File	1 KB
inst_mem.v	11/3/2019 10:38 AM	V File	7 KB
Readme.txt	1/9/2021 12:25 AM	TXT File	1 KB
top.v	5/6/2021 4:09 PM	V File	1 KB

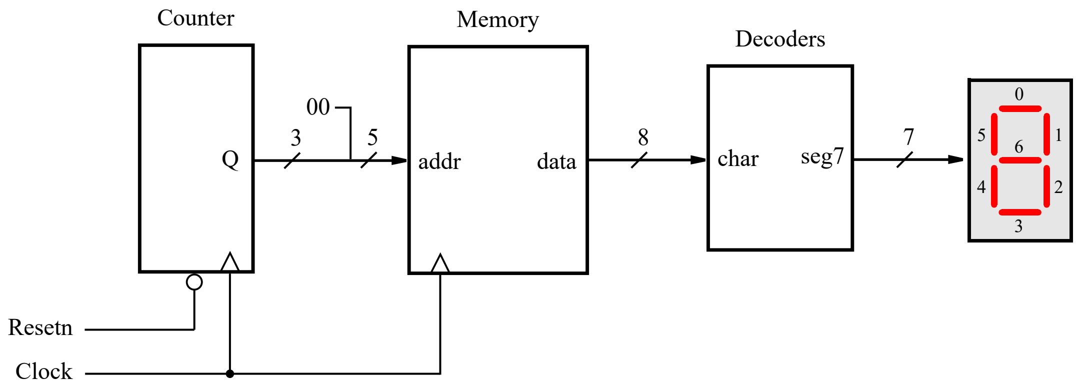
Figure 14: The *display* folder.

Figure 15 shows the Verilog code for *Display.v*, which has ports named *KEY*, *SW*, *HEX0*, and *LEDR*. Figure 16a gives a logic circuit that corresponds to the code in Figure 15. The circuit contains a counter that is used to read the contents of successive locations in a memory. This memory provides codes in ASCII format for some upper- and lower-case letters, which are provided as inputs to a decoder module. The counter and memory modules have a common clock signal, and the counter has a synchronous clear input. Each successive clock cycle advances the counter and reads a new ASCII code from the memory. Since the counter is three-bits wide, only the first eight locations in the memory are read (the upper two address bits on the memory are set to 00), and they provide the ASCII codes for letters A, b, C, d, E, F, g, and h. The decoder produces an appropriate bit-pattern to render each letter on a seven-segment display.

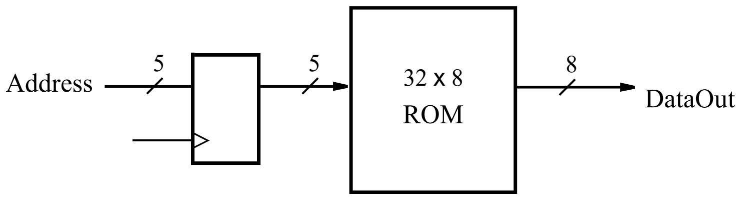
The memory used in the logic circuit is depicted in part b of Figure 16. It is a 32×8 synchronous read-only memory (ROM), which has a register for holding address values. The memory is specified in the Verilog file *inst_mem.v*, and it is initialized with the contents of the file *inst_mem.mif*, which is illustrated in Figure 17. This file contains the ASCII codes for the eight letters displayed by the circuit.

```
module Display (KEY, SW, HEX0, LEDR);
    input [0:0] KEY;
    input [0:0] SW;
    output reg [6:0] HEX0;
    output [9:0] LEDR;

    parameter A = 8'd65, b = 8'd98, C = 8'd67, d = 8'd100, E = 8'd69,
        F = 8'd70, g = 8'd103, h = 8'd104;
    wire Resetn, Clock;
    wire [2:0] Count;
    wire [7:0] char;

    assign Resetn = SW[0];
    assign Clock = KEY[0];

    count3 U1 (Resetn, Clock, Count);
    inst_mem U2 ({2'b0, Count}, Clock, char);
    assign LEDR = {2'b0, char};

    always @(*)
        case (char)
            A: HEX0 = 7'b0001000;
            b: HEX0 = 7'b0000011;
            C: HEX0 = 7'b1000110;
            d: HEX0 = 7'b0100001;
            E: HEX0 = 7'b0000110;
            F: HEX0 = 7'b0001110;
            g: HEX0 = 7'b0010000;
            h: HEX0 = 7'b0001011;
            default HEX0 = 7'b1111111;
        endcase
endmodule

module count3 (Resetn, Clock, Q);
    input Resetn, Clock;
    output reg [2:0] Q;

    always @ (posedge Clock)
        if (Resetn == 0)
            Q <= 3'b000;
        else
            Q <= Q + 1'b1;
endmodule
```

Figure 15: Verilog code for the *display* project.



a) circuit



b) memory module

Figure 16: A circuit that represents the *display* project.

```
DEPTH = 32;
WIDTH = 8;
ADDRESS_RADIX = HEX;
DATA_RADIX = DEC;
CONTENT
BEGIN
        00  : 65;        % A %
        01  : 98;        % b %
        02  : 67;        % C %
        03  : 100;       % d %
        04  : 69;        % E %
        05  : 70;        % F %
        06  : 103;       % g %
        07  : 104;       % h %
END;
```

Figure 17: The *inst_mem.mif* memory initialization file.

To compile the *display* project, in the *DESim* GUI click the `Compile Testbench` command. This command executes the project's *run_compile.bat* script, which is in its `sim` folder. This batch file is shown below:

```
1     if exist ..\inst_mem.mif (
2         copy /Y ..\inst_mem.mif .
3     )
4     if exist ..\inst_mem_bb.v (
5         del ..\inst_mem_bb.v
6     )
7     if exist work rmdir /S /Q work
8
9     vlib work
10    vlog ../tb/*.v
11    vlog ../*.v
```

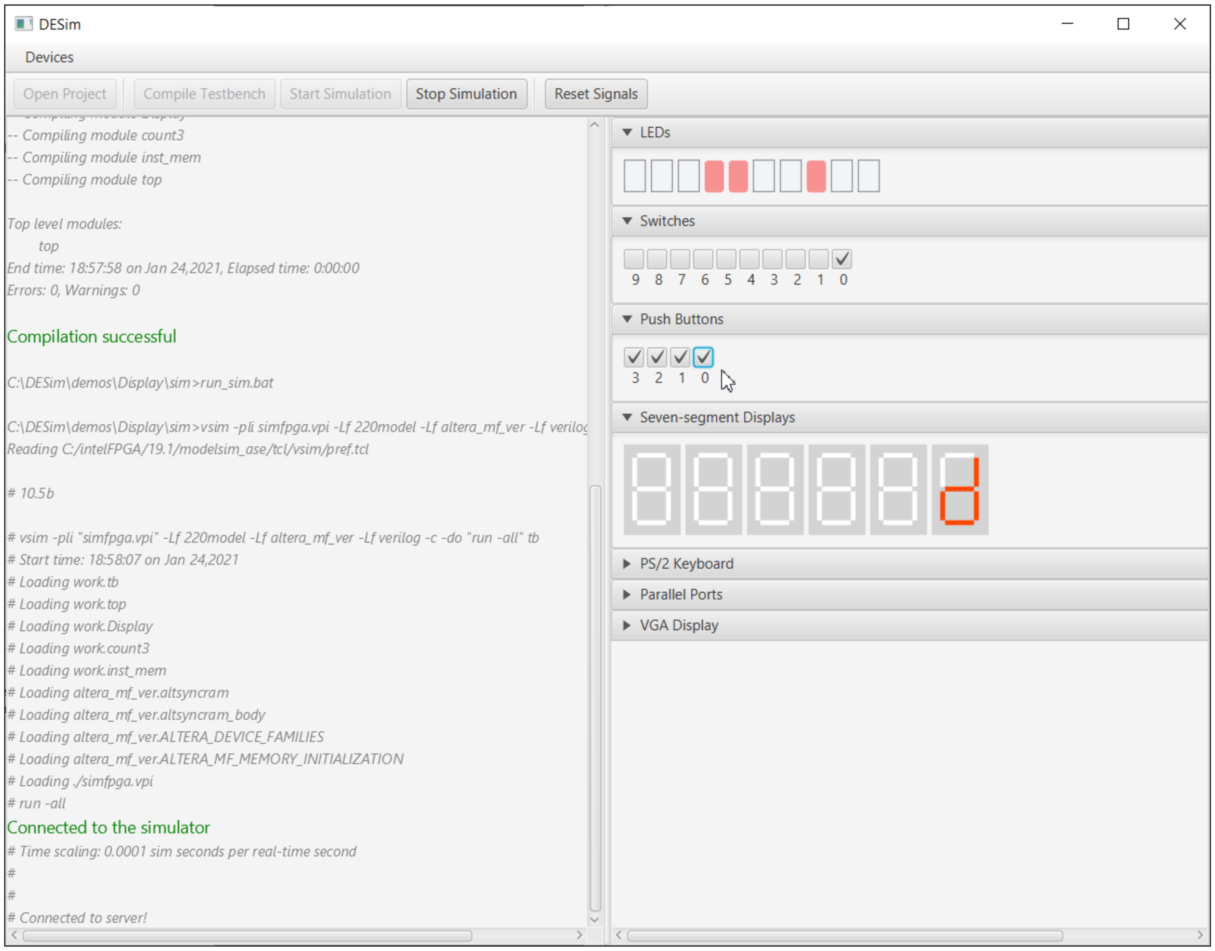
Lines 1 to 3 are used to copy the memory initialization file, *inst_mem.mif*, from the `display` project folder into the `sim` folder. This is done for two reasons: 1. *ModelSim* requires the file to be in the `sim` folder to properly initialize the memory module during a simulation, and 2. if the file is changed in the `display` folder, then the latest version of the file will always be used when starting a simulation. Lines 4 to 6 delete a file called *inst_mem_bb.mif* that is sometimes associated with a memory module; if present, this file would cause a *ModelSim* error. The rest of the batch file, which compiles the Verilog code, is the same as for the previously-described *DESim* projects.

The testbench file *tb.v* that is compiled by *run_compile.bat* for the *display* project is identical to the one used for the *addern* project, shown in Figure 7. To execute the testbench for the *display* project, click on `Start Simulation`. Its *run_sim.bat* script is the same as the ones used for the *addern* and *counter* projects.

The *Readme.txt* file for the *display* project is shown in Figure 18. You can follow its instructions to read successive locations out of the memory and display the corresponding characters on `HEX0`. An example simulation output after first resetting the circuit and then creating a few clock cycles using `KEY[0]` is illustrated in Figure 19.

```
To use this demo:

-- The clock input is created by toggling KEY[0]
-- The active-low synchronous reset input is SW[0]

The circuit displays "characters" stored in a ROM on HEX0.
To use the circuit:

1. Set SW[0] to 0 to allow the circuit to be reset
2. pulse KEY[0] down/up to make a clock cycle
    -- the character 'A', the first character stored
       in the ROM should be displayed on HEX0
3. Set SW[0] to 1 so that the reset is not active
4. pulse KEY[0] down/up to make a clock cycle
5. pulse KEY[0] down/up to make a clock cycle
    -- HEX0 should now show 'b', the next character
       stored in the ROM
6. pulse KEY[0] down/up to make a clock cycle
    -- HEX0 should now show 'C', the next character
       stored in the ROM
7. etc (there are eight characters stored in the ROM)
```

Figure 18: The *Readme.txt* file for the *display* project.



```
-- Compiling module count3
-- Compiling module inst_mem
-- Compiling module top

Top level modules:
      top
End time: 18:57:58 on Jan 24,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0

Compilation successful

C:\DESim\demos\Display\sim>run_sim.bat

C:\DESim\demos\Display\sim>vsim -pli simfpga.vpi -Lf 220model -Lf altera_mf_ver -Lf verilog
Reading C:/intelFPGA/19.1/modelsim_ase/tcl/vsim/pref.tcl

# 10.5b

# vsim -pli "simfpga.vpi" -Lf 220model -Lf altera_mf_ver -Lf verilog -c -do "run -all" tb
# Start time: 18:58:07 on Jan 24,2021
# Loading work.tb
# Loading work.top
# Loading work.Display
# Loading work.count3
# Loading work.inst_mem
# Loading altera_mf_ver.altsyncram
# Loading altera_mf_ver.altsyncram_body
# Loading altera_mf_ver.ALTERA_DEVICE_FAMILIES
# Loading altera_mf_ver.ALTERA_MF_MEMORY_INITIALIZATION
# Loading ./simfpga.vpi
# run -all
Connected to the simulator
# Time scaling: 0.0001 sim seconds per real-time second
#
#
# Connected to server!
```

Figure 19: Simulating the *display* project.

Setting up a *DESim* Project

An easy way to set up your own *DESim* project is to use one of the example projects in the *DESim* `demos` folder as a starting point. You should choose a specific `demo` project according to its features. For example, if your design project includes a memory module, then you might choose to start with a copy of the *display* project. But if you do not require a memory module, then you could start with a copy of one of the other projects. Also, you should start with a project that has the ports that you need in its "top" module that is instantiated by its testbench. All of the example projects described in this document have the same ports in their `top` module, but some other projects included in the `demos` folder may have different top-level ports.

Once you choose a project from the `demos` folder as a starting point, you should copy its folder contents into a new folder on your computer. For example, you might make a copy of `demos\display` and call the new folder `my_folder`. Then, in `my_folder` you would replace the file *Display.v* with your own source-code file, say *my_source.v*. Next, you would edit the file *top.v* in `my_folder` and change it to instantiate your Verilog module, say *my_module* (which would be in the file *my_source.v*). You would connect the signals in *top.v*, such as `CLOCK_50`, `KEY`, and so on, as needed to the ports of *my_module*. If some of the ports that you require for *my_module* aren't available in the *top* module, then you should instead use a different sample project from the `demos` folder that has the required ports in its *top* module.

You should not need to make any changes to the files in the `sim` or `tb` folder for your new project in `my_folder`. You can now open your new project in the *DESim* software and proceed to compile/simulate your code.

Troubleshooting Problems with the *DESim* Software

This section discusses some potential issues that could be encountered while using the *DESim* software, and provides suggested solutions.

1. Upon starting the *DESim* software you should see the message The server is running..." at the top of the *message pane* in the GUI. If you do not see this message, but instead see a message Server setup failed, then the *DESim* software is not working properly and should be closed. One reason why this would occur is if you have executed a *second* instance of the *DESim* program. The *DESim* software cannot be executed more than once concurrently on your computer.

2. If you click on the `Compile Project` command in the *DESim* GUI, it is possible to see an error message such as 'vlib' is not recognized as an internal or external command'. This error means that *DESim* attempted to execute the *vlib* program, which is part of the *ModelSim* software, but the program was not found by the operating system. This error will occur if the *ModelSim* software is not installed on the computer, or if it is installed but cannot be be located. There are two ways to fix the latter issue: 1) the Microsoft Windows `Path` environment variable can be updated to include the location of the *ModelSim* software, or 2) the location of the *ModelSim* software can be specified within the batch file that starts the *DESim* software. This batch file is called *DESim_run.bat* and is found in the file-system folder where *DESim* is installed. This second solution is the one that is used on the DESL and ECF systems. Hence, if you encounter this error on one of these computer systems, check that you are using the correct version of the *DESim* software; there is a different version for DESL and for ECF. Note that you can determine which system you are using by looking at the name of the `C:` drive on the computer.

3. Occasionally, when compiling or simulating a project in the *DESim* software you may see a *Warning* message which says that *ModelSim* cannot "unlink" a file. For example if your *DESim* project is stored in the folder C:\DESim\demos\addern, then this message would report:

   ```
   ** Warning: (vlog-31) Unable to unlink file "C:/DESim/demos/addern/sim/work/_lock"
   ```

   This problem occurs for unknown reasons and is caused by an issue with the *ModelSim* software (it happens when directly using the ModelSim GUI also, and not only when using the *DESim* tool). If the "unlink" issue persists (sometime it gets resolved automatically), then a solution is to browse with `File Explorer` into the file-system folder `C:\DESim\demos\addern\sim\work` and manually *delete* the file named *_lock*.