Using the DESim Application with Verilog Designs

For Quartus® Prime 24.1

1 Introduction

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 either the *ModelSim* or *Questa* simulation software. Using the *DESim* GUI you can invoke both the software's *HDL compiler* and *simulator*. Inputs to the 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:

- Getting Started with *DESim*
- Compiling and Simulating *DESim* Sample Projects
- Simulating a Circuit that Includes a Memory Module
- Making a *DESim* Project
- Troubleshooting Problems with the *DESim* Application

Requirements:

- A good working knowledge of Verilog hardware description language (HDL).
- A computer running Microsoft® Windows® (version 10 is recommended) or Ubuntu Linux.
- A computer running *ModelSim-Intel FPGA* edition software or *Questa-Intel FPGA* edition software
  - If using the *ModelSim-Intel FPGA Starter* edition software, version 10.5b is required. It is part of the *Quartus Prime* suite of CAD tools, versions 18.0, 18.1, 19.0 and 19.1, which are provided by Intel® Corporation.
  - If using the *Questa-Intel FPGA Starter* edition software, version 2021.2 is required. It is part of the *Quartus Prime* suite of CAD tools, version 21.1, which is provided by Intel® Corporation.
- You should know how to use the simulation software to simulate HDL code using a *testbench*. This material is presented in the tutorials: *Using ModelSim with Testbenches* and *Using Questa with Testbenches*, available from FPGAcademy.org.
- The *DESim* application. Instructions for downloading and installing the *DESim* application are available in the document *DESim Installation Guide*, available from `https://github.com/fpgacademy/DESim/releases`.

2 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 $\texttt{LEDR}_{9-0}$ that are provided on a DE1-SoC board. The `Switches` correspond to the board's $\texttt{SW}_{9-0}$ slide switches, the `Push Buttons` to $\texttt{KEY}_{3-0}$, and the `Seven-segment Displays` to `HEX5`, `HEX4`, ⋯, `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.

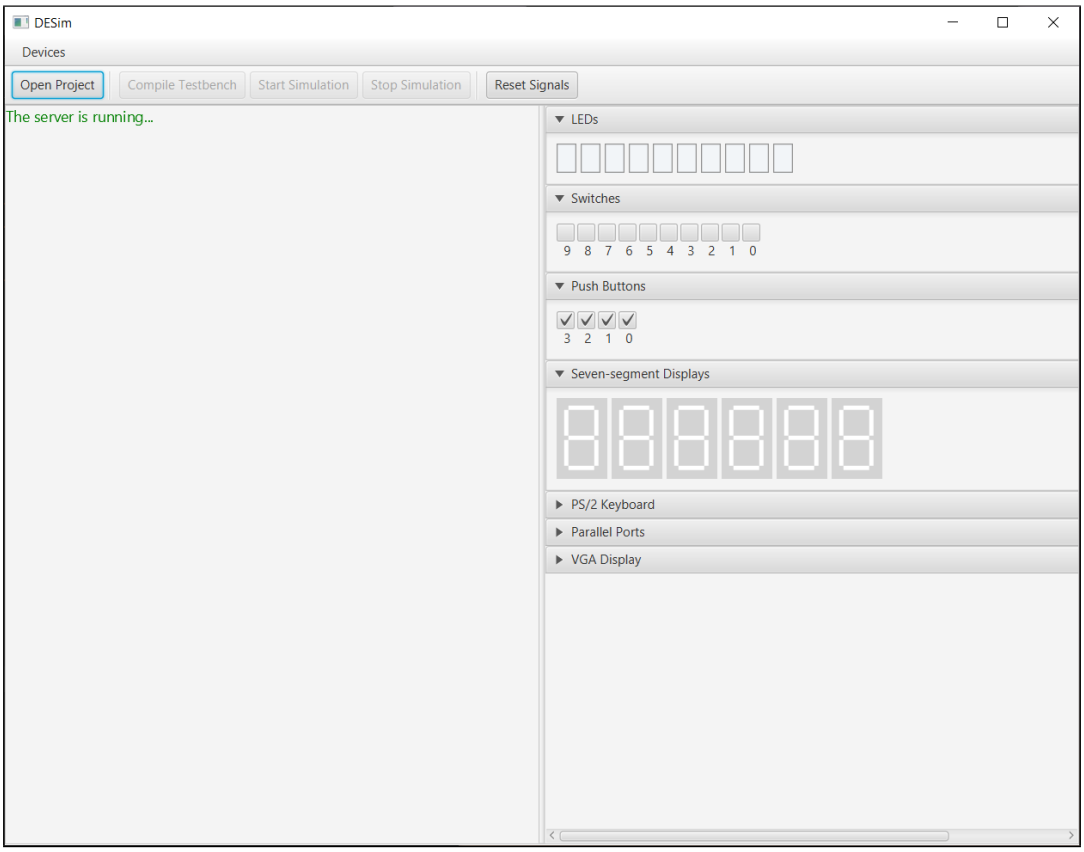

Figure 1. The *DESim* GUI.

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 that comes with the *DESim* software. Referring to Figure 1, click on the `Open Project` command to reach the dialogue displayed in Figure 2 (Windows) or Figure 3 (Linux). As shown in the figures, navigate to the `demos` folder, select either `modelsim/questa` and `systemverilog/verilog/vhdl`, click to select the `addern` project, and then click the `Select Folder` button in Windows or the `Open` button in Linux.

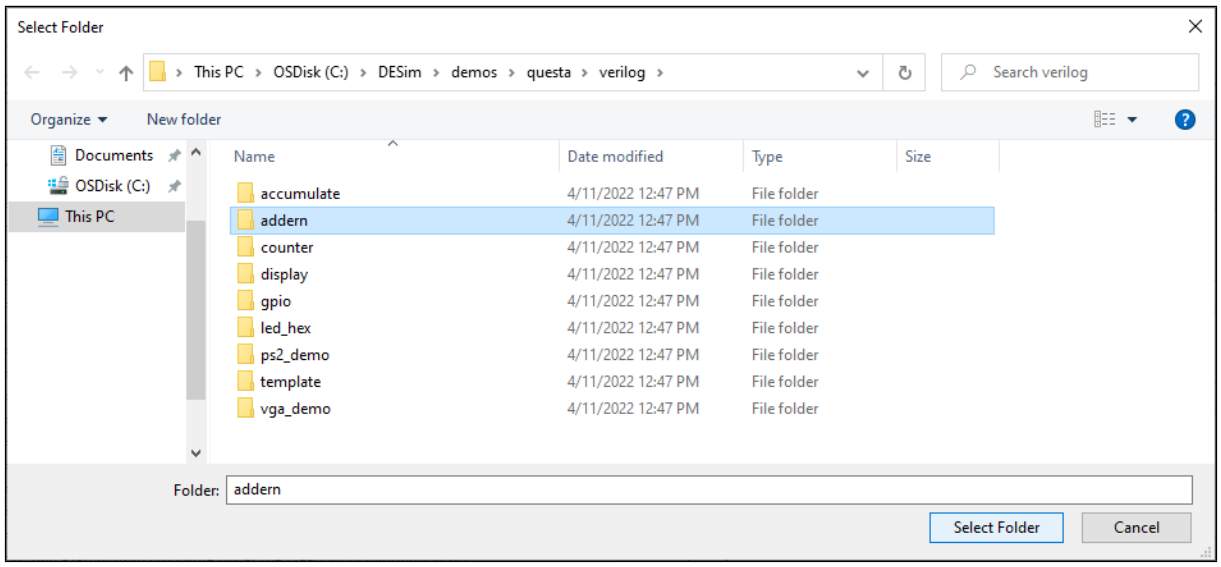

Figure 2. Opening the *addern* project under Windows.

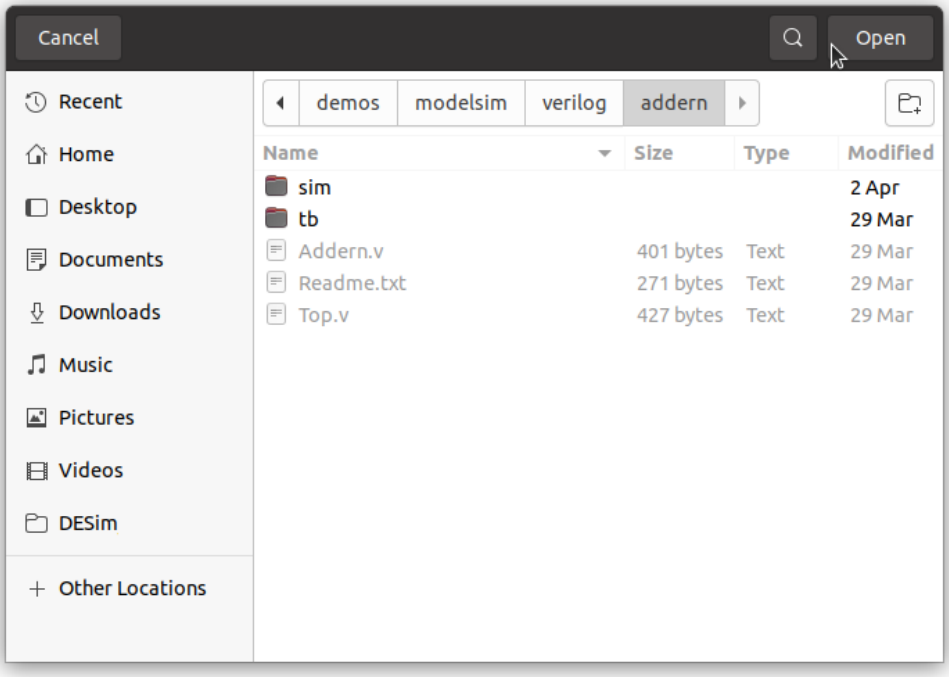

Figure 3. Opening the *addern* project under Linux.

The *addern* project folder, contains 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 a description of the functionality of this demo and is not really a part of the *DESim* project.

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 `LEDs`. 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 `KEY`, `SW`, and `LEDR`.

The *Addern* module is instantiated 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 $LEDR_4$.

```
module Addern (Cin, X, Y, Sum, Cout);
    parameter n = 4;

    input  wire         Cin;
    input  wire [n-1:0] X, Y;
    output wire [n-1:0] Sum;
    output wire         Cout;

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

endmodule
```

Figure 4. The Verilog source-code file `addern.v`.

```
module Top (KEY, SW, LEDR);
    input  wire [ 3: 0] KEY;        // DE-series pushbuttons
    input  wire [ 9: 0] SW;         // DE-series switches
    output wire [ 9: 0] LEDR;       // DE-series LEDs

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

endmodule
```

Figure 5. The Verilog source-code file `top.v`.

To compile the *addern* project in the *DESim* GUI, click the `Compile Testbench` command. This command executes a script called *run_compile*, which is found in the `sim` folder of the *addern* project. This script comprises some *ModelSim* commands. The *run_compile.bat* script for Windows is shown below (the *run_compile.sh* script for Linux is similar):

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

vlib work
vlog ../tb/*.v
if exist ../*.v (
    vlog ../*.v
)
if exist ../*.vhd (
    vcom ../*.vhd
)
```

The script executes the *vlib* command, which is part of the *ModelSim/Questa* software, to create a `work` folder (while first deleting this folder if it already exists). The script then invokes the simulation software's Verilog compiler, *vlog*, 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. If the SystemVerilog or Verilog version of the *addern* project was opened, the second call to *vlog* compiles the source-code of the *addern* project, which includes the files *Addern.v* and *top.v* or *Addern.sv* and *top.sv*. If instead the SystemVerilog or VHDL versions of the *Addern* project was opened, the call to *vcom* compiles the source-code of the *Addern* project, which includes the files *Addern.vhd* and *top.vhd*. Any messages produced while executing the *run_compile* script are displayed inside the message pane in the *DESim* GUI, as illustrated in Figure 6.

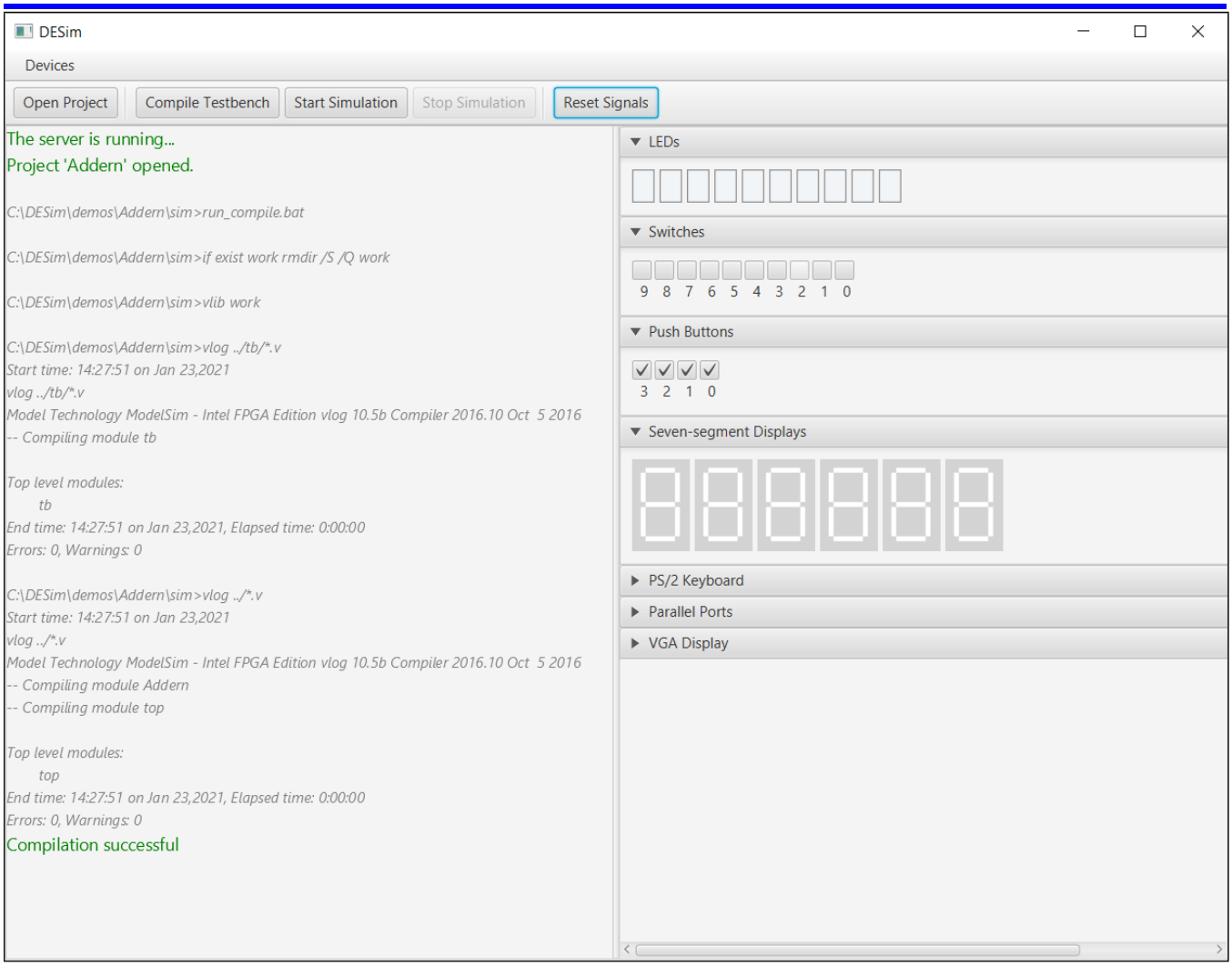

Figure 6. Messages produced by executing the *run_compile* script.

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 HDL code that might be used in various *DESim* projects. Hence, not all of the code in the testbench is needed for the *addern* project. The first line 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/Questa* 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 simulation software via the *sim_fpga* function. This capability is known as the *Verilog Procedural Interface* (VPI).

```
module tb();

    reg             CLOCK_50 = 0; // DE-series 50 MHz clock
    reg     [ 3: 0] KEY = 0;      // DE-series pushbutton keys
    reg     [ 9: 0] SW = 0;       // DE-series SW switches
    wire    [47: 0] HEX;          // HEX displays (six ports)
    wire    [ 9: 0] LEDR;         // DE-series LEDs

    reg             key_action = 0;
    reg     [ 7: 0] scan_code = 0;
    wire    [ 2: 0] ps2_lock_control;
    wire            ps2_clk;
    wire            ps2_dat;

    wire    [ 7: 0] VGA_X;        // "VGA" column
    wire    [ 6: 0] VGA_Y;        // "VGA" row
    wire    [ 2: 0] VGA_COLOR;    // "VGA pixel" colour (0-7)
    wire            plot;         // "Pixel" is drawn when this is pulsed
    wire    [31: 0] GPIO;         // DE-series 40-pin header

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

    // create the 50 MHz clock signal
    always #10
        CLOCK_50 <= ~CLOCK_50;
```

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

The second last line 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 simulator, click on the `Start Simulation` command in the *DESim* GUI. This command executes a script called *run_sim*, which is found in the `sim` folder of the *addern* project. This script runs *vsim*, the *ModelSim/Questa* Verilog simulator, using one of the commands:

```
vsim -pli simfpga.vpi -Lf 220model -Lf altera_mf_ver -Lf verilog -t 1ns -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. The `-t` argument specifies the simulator time resolution. Finally, the remaining arguments run the simulation for the top-level module, which is *tb*. Any messages produced while executing the *run_sim* script 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`.

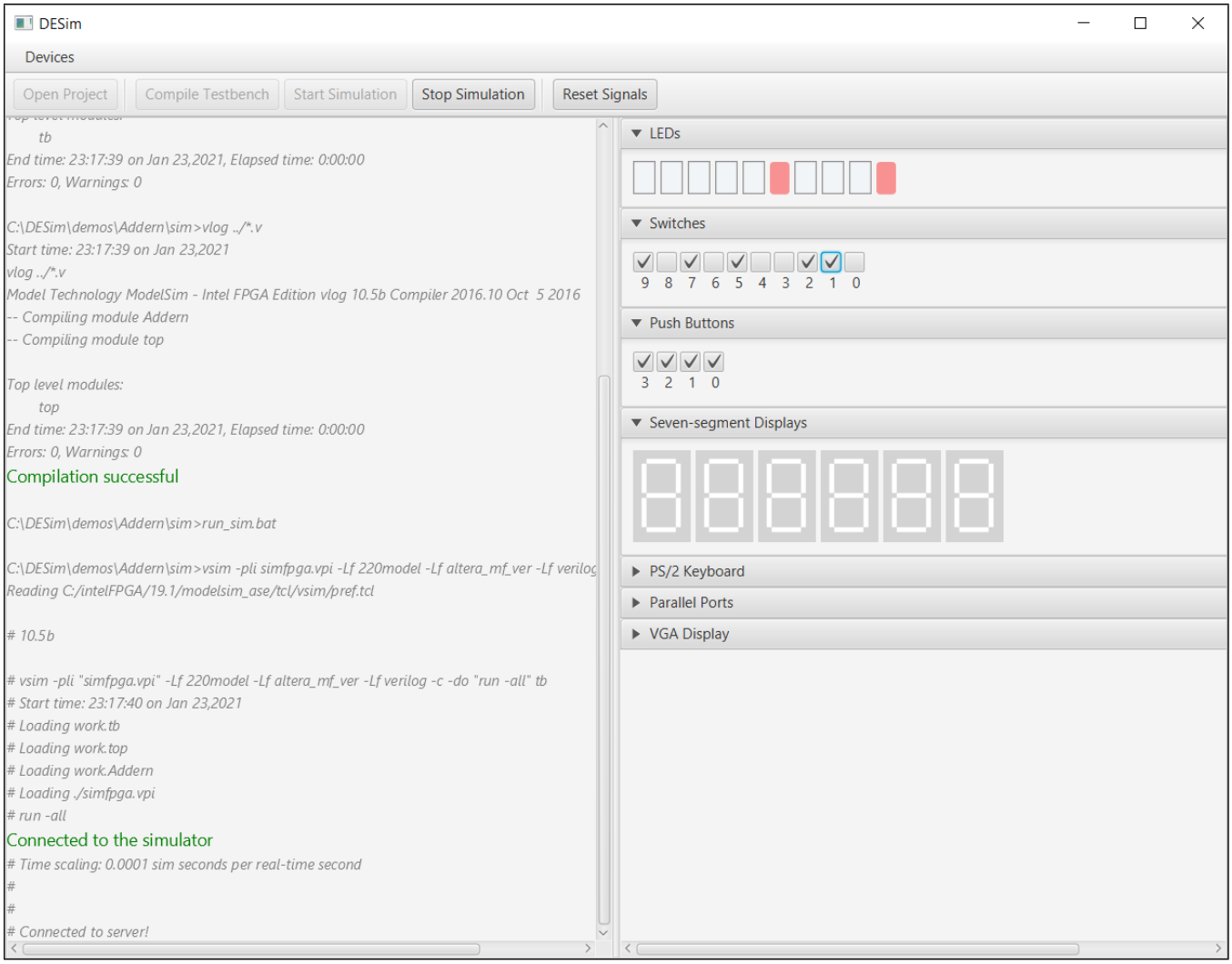

Figure 8. Messages produced by executing the *run_sim.* script.

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

In this demo:

-- X is the value of SW[3:0]
-- Y is the value of SW[7:4]
-- Cin is the value of SW[9]

To use:

1. various values to SW[9] and SW[7:0]
2. check the adder result on the LEDs
```

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

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

3 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. The contents of the file-system folder for this project look the same as for the *addern* project, except that there is a Verilog source-code file named *Counter.v*. Figure 10 shows the contents of the *Counter.v* file. It represents a 24-bit counter with synchronous reset. The *LEDR* port name corresponds to the red LEDs on the DE1-SoC board. Since *LEDR* port 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*.

As described for the *addern* project, the *Counter* module is *instantiated* in another Verilog module called *top*. This module is displayed in Figure 11. It is similar to the one from Figure 5, except that it includes a clock, *CLOCK_50*, input port and instantiates the *Counter* module. The port names for this module, correspond to the signal names on the DE1-SoC board, with *CLOCK_50* for the clock signal, and KEY_0 being used for reset.

To compile the *counter* project, in the *DESim* GUI click the `Compile Testbench` command. This command executes the *run_compile* script, which is found in the `sim` folder of the *counter* project. This script is identical to the one described earlier for the *addern* project. The testbench file that is compiled by *run_compile* for the *counter* project is found in its `tb` folder. This testbench, *tb.v*, is similar to the one shown in Figure 7, except that it includes a `CLOCK_50` signal in the DUT instantiation.

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

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

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

```
module Counter (CLOCK, RESETn, LEDR);
    input  wire         CLOCK;
    input  wire         RESETn;
    output wire [ 9: 0] LEDR;

    parameter n = 24;

    reg         [n-1:0] count;

    // 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 10. The Verilog code for the 24-bit counter.

```
module Top (CLOCK_50, KEY, LEDR);
    input  wire         CLOCK_50;   // DE-series 50 MHz clock signal
    input  wire [ 3: 0] KEY;        // DE-series pushbuttons
    output wire [ 9: 0] LEDR;       // DE-series LEDs

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

endmodule
```

Figure 11. The *top* module for the *counter* project.

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 12.

4 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, and how to initialize the stored contents of the memory in a *DESim* simulation. Use the `Open Project` command to open this example project. The contents of the file-system folder for this project look similar to the previous ones, but

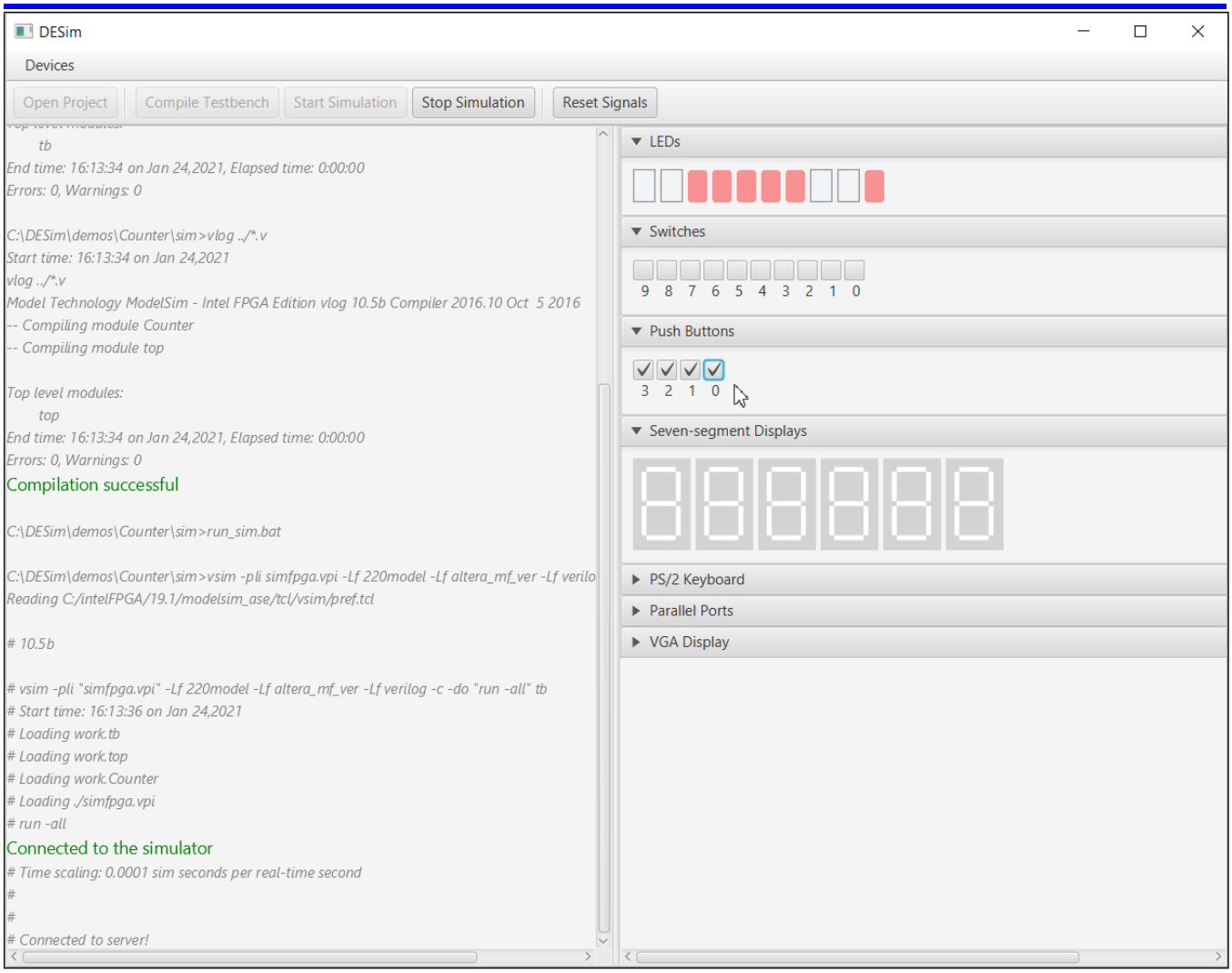

Figure 12. Simulating the *counter* project.

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.

Figure 13 shows the Verilog code for *Display.v*, which has ports named *CLOCK*, *RESETn*, *HEX0*, and *LEDR*. Figure 14*a* gives a logic circuit that corresponds to the code in Figure 13. 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 14. 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 15. This file contains the ASCII codes for the eight letters displayed by the circuit.

```
module Display (CLOCK, RESETn, HEX0, LEDR);
    input  wire         CLOCK;
    input  wire         RESETn;
    output reg  [ 6: 0] HEX0;
    output wire [ 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        [ 2: 0] count;
    wire        [ 7: 0] char;

    count3 U1 (CLOCK, RESETn, 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 (CLOCK, RESETn, Q);
    input  wire         CLOCK;
    input  wire         RESETn;
    output reg  [ 2: 0] Q;

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

Figure 13. The Verilog code for the *display* project.

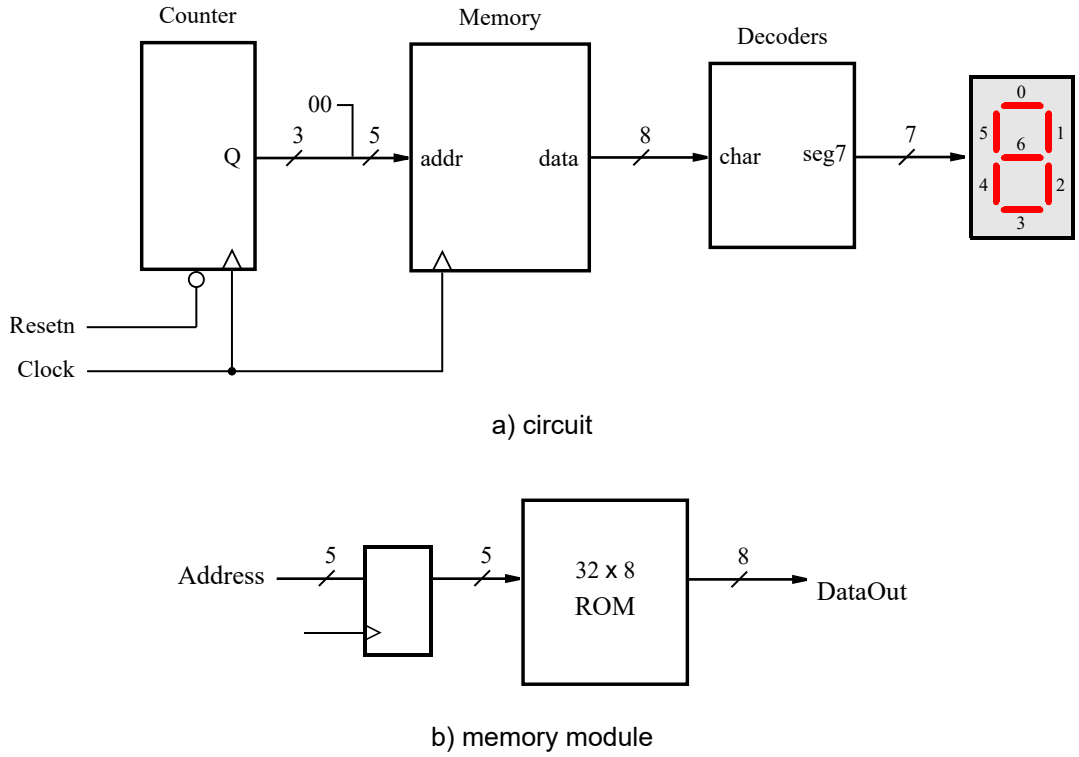

Figure 14. A circuit that represents the *display* project.

```
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 15. 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* script, which is in its `sim` folder. This *Windows* batch script is shown below (the *Linux* shell script is similar):

```
if exist ..\inst_mem.ip.mif (
    copy /Y ..\inst_mem.ip.mif .
)
if exist work rmdir /S /Q work

vlib work
vlog ../tb/*.v
vlog ../*.v
if exist ../*.vhd (
    vcom ../*.vhd
)
```

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/Questa* require 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. The rest of the script, which compiles the HDL code, is the same as for the previously-described *DESim* projects, except of the *vlog ../*.v* line. This line is now always executed, as the memory initialization module is written in Verilog, even if the SystemVerilog or VHDL versions were chosen.

The testbench file *tb.v* that is compiled by *run_compile* script for the *display* project is similar to the one used for the *addern* project, shown in Figure 7. The one difference, is the inclusion of the *CLOCK_50* signal in the `DUT` instantiation. To execute the testbench for the *display* project, click on `Start Simulation`. Its *run_sim* 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 16. 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 17.

5 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. The various 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\questa\verilog\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

```
The circuit displays "characters" stored in a ROM on HEX0.

In this demo:

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

To use:

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 16. The *Readme.txt* file for the *display* project.

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.

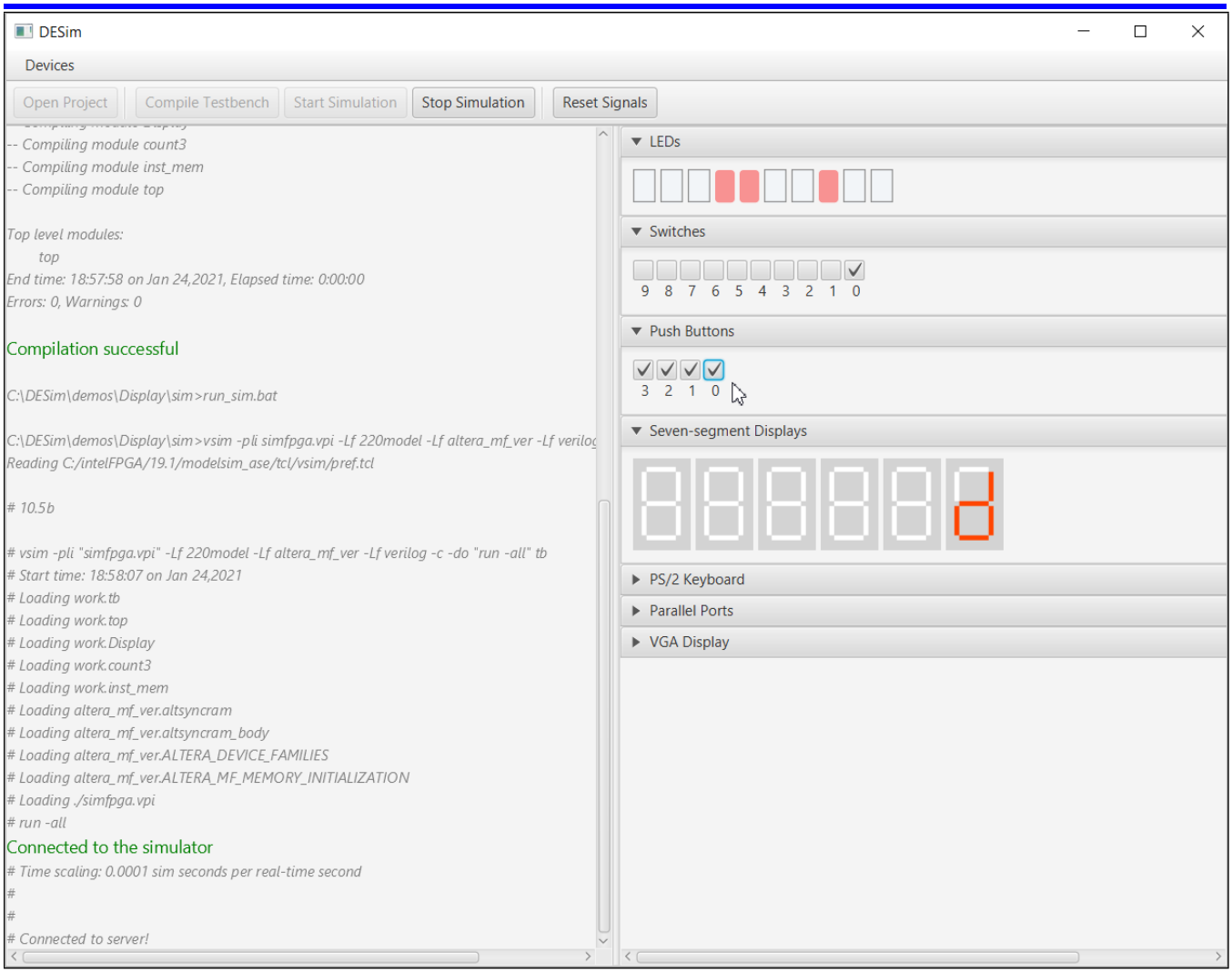

Figure 17. Simulating the *display* project.

6 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/Questa* software, but the program was not found by the operating system. This error will occur if the *ModelSim/Questa* 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`Path` environment variable can be updated to include the location of the *ModelSim/Questa* software, or 2) the location of the *ModelSim/Questa* software can be specified within the batch script that starts the *DESim* software. This script is called *DESim_run* and is found in the file-system folder where *DESim* is installed.

3. Occasionally, when compiling or simulating a project in the *DESim* software you may see a *Warning* message which says that *ModelSim/Questa* 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/Questa* software (it happens when directly using the ModelSim/Questa 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*.

4. If you click on the `Start Simulation` command in the *DESim* GUI, an error message such as Load of "./simfpga.vpi" failed: Bad DLL Format might happen. This error occurs if the *./simfpga.vpi* DLL is not compatible with the installed simulator. Make sure that you are using the corresponding *./simfpga.vpi* DLL generated for *ModelSim* when using the *ModelSim* simulator, and the one generated for *Questa* when using the *Questa* simulator. They can be found in the corresponding demo directories. If the error is still present, verify that the correct version of the *ModelSim/Questa* software is been used. Refer to the *DESim* Installation Guide for supported versions.