EEC180B Lab 3 - Synopsys Tutorial

University of California, Davis
Department of Electrical and Computer Engineering

Objective: The purpose of this tutorial is to introduce the Synopsys simulation and synthesis tools. A simple VHDL description of an up/down binary counter will be functionally simulated. The design will then be synthesized into generic gates using an example "class" library or into a Xilinx field programmable gate array (FPGA). After using Xilinx software to process the Synopsys output file, the design can be downloaded to a Xilinx FPGA and tested. An exercise is provided at the end of the tutorial to test your understanding of the high-level design process.

I. Setting up the environment

First set up your environment for the Synopsys and Xilinx tools. To do this, run the setup script by typing the following command:

```
setup synopsys9802
```

If your environment isn't set up for the Xilinx tools, run the setup script for Xilinx:

```
setup xilinx
```

These setup scripts just need to be run once. Open a new window or log out and log back in to get the new environment. After getting a new window, type the following commands to verify that your configuration:

```
echo $XILINX

echo $SYNOPSYS
```

These commands should display the UNIX paths to the Xilinx and Synopsys root directories, respectively.

Now create a new directory for the tutorial exercises and copy the tutorial files into it. Once you have changed directory to your new directory, use the following commands where the final period is the UNIX designation for your current directory:

```
cp /afs/ece/classes/eec180b/xtutorial/.sy*  .

cp /afs/ece/classes/eec180b/xtutorial/*.*   .
```

The .synopsys_vss.setup and the .synopsys_dc.setup files specify the default working directory as ./WORK, which means that you must create a WORK subdirectory under your current directory.

```
mkdir WORK
```

II. On-line documentation

The Synopsys on-line documentation is available by typing the command:

```
sold
```

The Simulation Tools and the Synthesis Tools collections have a number of useful references. Each collection also has a set of documents, which are formatted for printing such as the Simulation Documents for Printing or the Synthesis Documents for Printing. The VSS Quick Start tutorial, which is in the Simulation Tools collection, is a useful guide to the Synopsys simulator. (It is available as a print-formatted version as well as a version for on-line use.)

The Xilinx documentation is also available on-line using the command:

```
dtext &
```

The Synopsys (XSI) Interface/Tutorial Guide is a useful reference for this course.

III. Functional Simulation

To simulate the updown counter design, you must first compile, or *analyze*, the VHDL source files to generate files which the VHDL Simulator can use. The updown design consists of two files, my_pkg.vhd and updown.vhd, which must both be analyzed using the commands shown below. Since my_pkg is referenced in the updown design file, my_pkg.vhd should be analyzed first.

```
vhdlan  -i my_pkg.vhd

vhdlan  -i updown.vhd
```

You will use a testbench VHDL file, tb.vhd, to provide part of the simulation inputs. You must analyze this file also so that it can be simulated. However, this file will not be synthesized.

```
vhdlan  -i tb.vhd
```

Now you are ready to simulate the counter. The Synopsys VHDL Debugger, vhdldbx, can be used for interactive simulation sessions. Run the following command:

```
vhdldbx &
```

The design, which we want to simulate, is the *configuration* of the testbench program. Select TB_CFG in the design window and click on the OK box to bring up the VHDL Debugger.

To trace all the signals in the top-level design file (i.e. the testbench), type the following at the Command prompt (#):

```
tr *’signal
```

You should also trace the count signal, which is declared in the updown design. You can change the "working region" by typing:

```
cd uut
```

In the VHDL Debugger source code region, double-click on the signal name count and click on the *Trace* button. Alternatively, you could type **tr count** at the Command prompt. A third method would be to type **tr uut/count** from the original working region.

The trace commands should bring up a Waveform Viewer window. Since our stimulus is partially coded into the testbench file, we can simply run for some amount of time and observe the signal waveforms.

```
run 200
```

The count value should be cleared to 0 since the reset signal was low for a clock cycle. Notice that the enable and up_dn signals are undefined since the testbench program does not initialize these signals. To assign a logical 0 to these signals, type the following at the Command prompt:

```
as ’0’ enable up_dn

run 200
```

The enable and up_dn signals should be low in the Waveform Viewer. Since the enable signal is active-high, the count remains at 0. To enable counting, use the following commands:

```
as ’1’ enable

run 600
```

You can continue entering commands at the Command prompt and testing the counter as desired. The reset signal can also be tested using the assign command:

as ’0’ reset

run 200

as ’1’ reset

Another way to enter simulation commands is through a simulation command file. Close the Waveform Viewer window by selecting File -> Exit. Then Choose Execute -> Restart from the Vhdldbx menu bar in order to restart your simulation. Delete the Arguments field in the dialog box and again select the TB_CFG design with the DEFAULT library. Run a simulation by typing the following command at the Command prompt:

include sim.script

Verify the simulation and make sure you understand the sim.script commands. The sim.script file is an ASCII text file so you can use vi or another editor to view or modify the script. Modify the simulation script to fully test the counter.

Occasionally, the Synopsys Waveform Viewer will not display the waveforms even though the correct simulation commands have been given. Check your directory for any waveform files (the final suffix of the file will be .ow). You can use File -> Open from the Waveform Viewer pull-down menus to view the waveform stored in the file. This file can also be safely deleted when you are done viewing it.

IV. Synthesis

Part A - Class library

In this section, you will synthesize the design to generic gates using an example "class" library. You must set the .synopsys_dc.setup file to point to the correct target libraries. There are two example .synopsys_dc.setup files included in the tutorial files. To target the class library, type

copy .synopsys_dc.setup.class .synopsys_dc.setup

The Synopsys tools which are used for synthesis are the Design Compiler or the Design Analyzer. In order to process a design interactively, you can use the Design Analyzer. In many cases, however, it is more efficient to write a Design Compiler script and process the design in batch mode.

You can run the Synopsys Design Analyzer program by typing:

design_analyzer &

You could then synthesize your design by using the pull-down menu options or by typing commands at the Command Window prompt. However, the best way to synthesize a design is by using a script file.

Open the Command Window from the Setup pull-down menu. This will allow you to see the results of command execution. To execute the class.scr script, select the Execute Script item from the Setup menu and click on class.scr. Once the script has completed, you can view the gate-level representation of your synthesized design. Select the updown module and click the down-arrow button and then the AND gate icon on the left side of the Design Analyzer window.

Another alternative, which gives the fastest run-time, is to run the script in batch mode without graphically displaying the results. To run a Design Compiler script in batch mode, use the following command:

```
dc_shell  -f  class.scr > class.log &
```

If the file class.log already exists, you will need to delete or rename it before running the preceeding command. This command will take several minutes. You can check the progress of the synthesis by displaying the log file using commands such as "more", "less", "tail", "cat", etc.

```
tail class.log
```

The script file does not set optimization constraints for timing or area. To check the synthesis results, you can examine the report files, updown_class.area and updown_class.timing. By setting optimization goals before compiling the design, you could affect the speed or size of the final synthesized circuit. This is beyond the scope of this tutorial, however. Synopsys has a number of on-line resources which cover design optimization.

To view or print the gate-level schematic, run the design_analyzer and read in the updown_class.db file. Select the updown module and click on the down-arrow button and then the AND gate icon on the left edge of the Design Analyzer window to view the schematic.

Part B - Xilinx library

To set your .synopsys_dc.setup file to target the Xilinx libraries, type

```
copy .synopsys_dc.setup.xilinx  .synopsys_dc.setup
```

To synthesize the counter using Xilinx libraries, type

```
dc_shell  -f  updown.scr > updown.log &
```

If the file updown.log already exists, you will need to delete or rename it before running the preceeding command. The synthesis script will take several minutes to complete. As mentioned earlier, you can check the progress of the script by checking the log file such as with the following command:

```
more updown.log
```

The result of the synthesis script will be an output file in sxnf (Synopsys Xilinx Netlist Format) format.

Now the design is ready to be implemented in a Xilinx FPGA using the Xilinx design tools. Open the Xilinx design manager by typing the command:

```
dsgnmgr &
```

Select **New Project...** from under the **File** pull-down menu. Using the Browse button, select **updown.sxnf** as the input design. A default work sub-directory will be created for you. Click on the OK button to accept the inputs and close the dialog box.

Select **Implement...** from under the **Design** pull-down menu. You will be implementing the updown counter design in a Xilinx XC4005E-3-PC84. Since we specified this part in the Synopsys synthesis script, this will be the default part in the dialog box which appears. Therefore, you do not need to set the part number. Simply click on the **Run** button process your design. This will take a few minutes, but you will be able to observe the progress on the graphical user interface.

Once the design is processed, you can use the **Report Browser** under the **Utilities** pull-down menu to view the various report which the Xilinx tools generate. For example, you can check the Pad Report to verify that the pad locations have been assigned as specified in the updown.attr file. The Post Layout Timing Report contains interesting information such as the maximum frequency at which your design can operate. Explore the other reports to find out how many FPGA resources (CLBs and IOBs) were used by the design implementation.

V. Downloading to the Xilinx demo board

The final bit file, updown.bit, can be downloaded to a Xilinx demo board and verified. You must transfer the bit file to one of the PCs in room 2112 by either a **binary** mode FTP session from a DOS window or by logging in to the file server so that your workstation home directory is mounted as a DOS drive. (If you log in so that your workstation home directory is mounted as a DOS drive, **remember to log out when you are done so that others don't gain access to your files!**)

Create a directory for yourself on a PC and copy or binary FTP your file into your new directory.

Next, set up the Xilinx demo board.

- The Xilinx demo board should be attached to the serial port using the 9-pin Xchecker cable.

- The Xilinx board operates on **+5V** DC. Make sure to check and correctly set the power supply voltage **before** connecting it to the Xilinx board and turning it on. Don't assume that the power supply will be set to +5V.

- You will need to provide a **0 to +5V** square wave for the clock signal which has been assigned to pin P72. Make sure to check the voltage using the oscilloscope **before** connecting the function generator output to the Xilinx board.

- Connect the power supply to the Xilinx board and turn it on.

To download the bit file to the Xilinx demo board, type

```
xchecker updown.bit
```

The xchecker program will automatically detect the Xilinx board attached to the serial port and prompt you to enter a carriage return to begin downloading. Once the design has been downloaded, turn on the function generator and use the switches to verify that the design works as intended.

VI. Exercise (Optional)

Modify the counter example such that it becomes a two-digit **BCD** up/down counter instead of a binary up/down counter. Your counter must have a range of 00 to 99 and the output display must be in decimal format. The counter should automatically rollover from 99 to 00 when counting up or from 00 to 99 when counting down. Simulate, synthesize and download your new design. Demonstrate your design to the TA.

Hint: Define two 4-bit signals for the most-significant and least-significant digits instead of a single 8-bit count signal. Use if-then statements to specify the logic for your counter.

VII. Appendix A - Source files, Setup file, Scripts

my_pkg.vhd

```
-- Package for updown counter example
-- The function hex_7seg takes a 4-bit hex input and produces
-- the corresponding 7-segment display driver signals, which
-- are active low.  The segs (0 to 6) correspond to segments
-- A, B, C, D, E, F, G, respectively.

Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

package my_pkg is
subtype hex is std_logic_vector(3 downto 0);
subtype segs is std_logic_vector(0 to 6);
constant zero : std_logic_vector(7 downto 0) := "00000000";
function hex_7seg (hexval : hex) return segs;
end my_pkg;

package body my_pkg is
function hex_7seg (hexval : hex) return segs is
begin
   case hexval is
         when "0000" => return ("0000001");
         when "0001" => return ("1001111");
         when "0010" => return ("0010010");
         when "0011" => return ("0000110");
         when "0100" => return ("1001100");
         when "0101" => return ("0100100");
         when "0110" => return ("0100000");
         when "0111" => return ("0001111");
         when "1000" => return ("0000000");
         when "1001" => return ("0001100");
         when "1010" => return ("0001000");
         when "1011" => return ("1100000");
         when "1100" => return ("0110001");
         when "1101" => return ("1000010");
         when "1110" => return ("0110000");
         when "1111" => return ("0111000");
         when others => return ("-------");
   end case;
end; -- function
end my_pkg;
```

updown.vhd

```
-- Updown counter example.
-- Displays binary count on 2-digit hexadecimal display.
-- Counts up/down in range of 00 to FF.

Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use WORK.my_pkg.all;

entity updown is
        port (clk, reset, enable, up_dn : in std_logic;
      msb_seg, lsb_seg : out segs);
end updown;

architecture behave of updown is
signal count : std_logic_vector(7 downto 0);
begin

COUNTER : process
begin
wait until clk'event and clk='1';
if (reset='0') then                -- reset on Xilinx board active low
        count <= zero;             -- synchronous reset
elsif (enable='1') then
        if (up_dn='1') then
                count <= count+1;
        else
                count <= count-1;
        end if;
end if;
end process;

msb_seg <= hex_7seg(count(7 downto 4));   -- use hex_7seg function
lsb_seg <= hex_7seg(count(3 downto 0));

end behave; -- architecture

configuration updown_cfg of updown is
for behave
end for;
end updown_cfg;
```

tb.vhd

```
-- testbench for updown counter example
-- Generates the clk signal using a process.
-- Initializes the reset signal using a process.
-- up_dn and enable are not set. These can be controlled
-- manually during simulation.

Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use WORK.my_pkg.all;

entity tb is
end tb;

architecture behave of tb is
signal clk, reset, enable, up_dn : std_logic;
signal msb_seg, lsb_seg : segs;

component updown
        port (clk, reset, enable, up_dn : in std_logic;
                msb_seg, lsb_seg : out segs);
end component;

begin

uut : updown
 port map(clk, reset, enable, up_dn, msb_seg, lsb_seg);

initial: process
begin
        reset <= ’0’;
        wait for 100 ns;
        reset <= ’1’;
        wait;                   -- process hangs here forever
end process;

clkgen : process
begin
        clk <= ’0’;
        wait for 50 ns;
        clk <= ’1’;
        wait for 50 ns;
end process;

end behave;
```

```
configuration tb_cfg of tb is
for behave
end for;
end tb_cfg;
```

sim.script

```
tr *'signal
tr uut/count
run 200
as '1' enable up_dn
run 25600
```

.synopsys_vss.setup

```
-- ==================================================== --
-- Template .synopsys_vss.setup file for Xilinx design --
--             For use with Synopsys VSS.             --
-- ==================================================== --

-- ==================================================== --
--           Set any simulation preferences.          --
-- ==================================================== --
TIMEBASE        = NS
TIME_RES_FACTOR = 1

-- ================================================== --
-- Define a work library in the current project dir   --
-- to hold temporary files and keep the project area  --
-- uncluttered. Note: You must create a subdirectory  --
-- in your project directory called WORK.             --
-- ================================================== --
WORK    > DEFAULT
DEFAULT : ./WORK

-- ================================================== --
-- Note that the following simulation libraries are   --
-- provided ready-analyzed with VSS v9701. If you're  --
-- using a later version of VSS then refer to the     --
-- automatic compile scripts provided in the          --
-- appropriate library's source directory, e.g.       --
-- $XILINX/synopsys/libraries/sim/src/unisims         --
-- ================================================== --
UNISIM : $XILINX/synopsys/libraries/sim/lib/unisims
```

```
-- ================================================= --
-- VITAL SimPrim libraries provided to support back-  --
-- annotated simulation only.                         --
-- ================================================= --
SIMPRIM : $XILINX/synopsys/libraries/sim/lib/simprims

-- ================================================= --
-- Packages used by LogiBLOX functional simulation    --
-- models only. I.e. to support behavioral simulation --
-- of VHDL designs with instantiated LogiBLOX cells.  --
-- ================================================= --
LOGIBLOX : $XILINX/synopsys/libraries/sim/lib/logiblox

-- ================================================= --
-- Xilinx XC9000 FTGS simulation libraries.           --
-- ================================================= --
XC9000 : $XILINX/synopsys/libraries/sim/lib/xc9000/ftgs
```

.synopsys_dc.setup.class

```
search_path = {} + search_path
link_library = {class.db};
target_library = {class.db};
symbol_library = {class.sdb};
define_design_lib WORK -path WORK;
```

.synopsys_dc.setup.xilinx

```
/* ================================================= */
/* Template .synopsys_dc.setup file for Xilinx designs */
/*      For use with Synopsys FPGA Compiler.          */
/* ================================================= */

/* ================================================= */
/* The Synopsys search path should be set to point   */
/* to the directories that contain the various       */
/* synthesis libraries used by FPGA Compiler during  */
/* synthesis.                                        */
/* ================================================= */

XilinxInstall = get_unix_variable(XILINX);
SynopsysInstall = get_unix_variable(SYNOPSYS);

search_path = { .        \
        XilinxInstall + /synopsys/libraries/syn \
        SynopsysInstall + /libraries/syn }
```

```
          /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */
          /* Ensure that your UNIX environment */
          /* includes the two environment var- */
          /* iables: $XILINX (points to the    */
          /* Xilinx installation directory) and*/
          /* $SYNOPSYS (points to the Synopsys */
          /* installation directory.)          */
          /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */

/* ==================================================== */
/* Define a work library in the current project dir  */
/* to hold temporary files and keep the project area */
/* uncluttered. Note: You must create a subdirectory */
/* in your project directory called WORK.            */
/* ==================================================== */

  define_design_lib WORK -path ./WORK

/* ==================================================== */
/* General configuration settings.                  */
/* ==================================================== */

set_fix_multiple_port_nets -outputs

xnfout_constraints_per_endpoint = 0
xnfout_library_version = "2.0.0"

bus_naming_style = "%s<%d>"
bus_dimension_separator_style = "><"
bus_inference_style = "%s<%d>"

/* ==================================================== */
/* Set the link, target and synthetic library        */
/* variables. Use synlibs (with the -fc switch) to   */
/* determine the link and target library settings.   */
/* You may like to copy this file to your project    */
/* directory, rename it ".synopsys_dc.setup" and     */
/* append the output of synlibs. For example:        */
/*     synlibs -fc 4028ex-3 >> .synopsys_dc.setup    */
/* ==================================================== */

link_library = {xprim_4005e-3.db xprim_4000e-3.db xgen_4000e.db xfpga_4000e-3.db
xio_4000e-3.db}
target_library = {xprim_4005e-3.db xprim_4000e-3.db xgen_4000e.db xfpga_4000e-
3.db xio_4000e-3.db}
define_design_lib xdw_4000e -path /usr/pkg/xactm1.5/synopsys/libraries/dw/lib/xc4000e
symbol_library = {xc4000e.sdb}
synthetic_library = {xdw_4000e.sldb standard.sldb}
```

class.scr

```
/* ============================================================ */
/* CLASS library synthesis script file
/* ============================================================ */

/* To execute this DC Shell script, from command line, type:        */
/*        dc_shell -f class.scr > class.log &                       */

pkg_name=my_pkg.vhd
design_name=updown.vhd
ent_name=updown
arch_name=behave
output_name=updown_class

remove_design -all
analyze -format vhdl -lib DEFAULT {pkg_name, design_name}
elaborate ent_name -arch arch_name -lib DEFAULT -update
current_design ent_name
link
set_operating_conditions -library "class" "WCCOM"
set_wire_load  "10x10" -library "class"
compile  -map_effort medium
write -format db -hierarchy -output output_name + ".db"
report_area > output_name + ".area"
report_timing > output_name + ".timing"
quit
```

updown.scr

```
/* ==================================================*/
/*    Sample Script for Synopsys to Xilinx Using     */
/*                 FPGA Compiler                     */
/*                                                   */
/*  Targets the Xilinx XC4005E-3 and assumes a VHDL  */
/*        source file by way of an example.          */
/*                                                   */
/*   For general use with XC4000E/EX architectures.  */
/*      Not suitable for use with XC3000A/XC5200     */
/*                 architectures.                    */
/*                                                   */
/*   To run: dc_shell -f updown.scr > updown.log &   */
/*       or  dc_shell -f updown.scr | tee updown.log */
/*                                                   */
/* ==================================================*/

/* ================================================== */
/* Set the name of the design's top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                        */
/* ================================================== */

   PKG = my_pkg
   TOP = updown
                    /* ========================== */
                    /* Note: Assumes design file- */
                    /* name and entity name are   */
                    /* the same (minus extension) */
                    /* ========================== */

   part     = "4005epc84-3"

/* ================================================== */
/* Analyze and Elaborate the design file and specify */
/* the design file format.                           */
/* ================================================== */

   analyze -format vhdl PKG + ".vhd"
   analyze -format vhdl TOP + ".vhd"

                  /* ============================ */
                  /* You must analyze lower-level */
                  /* hierarchy modules here       */
                  /* ============================ */
   elaborate TOP
```

```
/* ================================================ */
/* Set the current design to the top level.         */
/* ================================================ */

   current_design TOP

/* ================================================ */
/* Set the synthesis design constraints.            */
/* ================================================ */

   remove_constraint -all

/* ================================================ */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O     */
/* synthesis.                                       */
/* ================================================ */

   set_port_is_pad "*"

   /* Some example I/O parameters */
   set_pad_type -no_clock all_inputs()

                     /* ========================== */
                     /* Note: Assumes clock port   */
                     /* named clk                  */
                     /* ========================== */

   set_pad_type -clock clk
   set_pad_type -slewrate HIGH all_outputs()

                  /* ============================ */
                  /* Note: Synopsys slew-control=  */
                  /* HIGH is the same as Xilinx’s  */
                  /* slewrate=SLOW. Synopsys slew- */
                  /* control=LOW is same as Xilinx */
                  /* slewrate=FAST.                */
                  /* ============================ */
   insert_pads

   uniquify          /* make duplicate modules unique */

/* ================================================ */
/* Synthesize and optimize the design               */
/* ================================================ */
```

```
   compile -boundary_optimization

/* ============================================ */
/* Write the design report files.                  */
/* ============================================ */

   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing"

/* ============================================ */
/* Write out the design to a DB file. (Post compile) */
/* ============================================ */

   write -format db -hierarchy -output TOP + "_compiled.db"

/* ============================================ */
/* Replace CLBs and IOBs with gates.                */
/* ============================================ */

   replace_fpga

/* ============================================ */
/* Set the part type for the output netlist.          */
/* ============================================ */

   set_attribute TOP "part" -type string part

/* ============================================ */
/* Optional attribute to remove the FPGA Compiler’s  */
/* mapping structures from the design. This permits  */
/* The Xilinx design implementation tools to map the */
/* design instead.                                 */
/* ============================================ */

/* set_attribute find(design,"*") "xnfout_write_map_symbols" \
     -type boolean FALSE  */

/* ============================================ */
/* Add any I/O constraints to the design.           */
/* ============================================ */

                  /* ========================= */
                  /* Note: Assumes design file- */
                  /* name and attribute file-   */
                  /* name are the same (minus   */
                  /* extension)                 */
                  /* ========================= */
   include TOP + ".attr"

/* ================================================== */
/* Save design in XNF format as <design>.sxnf        */
/* ================================================== */

   ungroup -all -flatten
   write -format xnf -hierarchy -output TOP + ".sxnf"

/* ================================================== */
/* Write out the design to a DB. (Post replace_fpga) */
/* ================================================== */

   write -format db -hierarchy -output TOP + ".db"

/* ================================================== */
/* Write-out the timing constraints that were        */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                     */
/* ================================================== */

   write_script > TOP + ".dc"

/* ================================================== */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view        */
/* dc2ncf.log to review the translation process.     */
/* ================================================== */

/*   sh dc2ncf TOP + ".dc"                            */

/* ================================================== */
/* Exit the Compiler.                                */
/* ================================================== */

   exit

/* ================================================== */
/* Now run the Xilinx design implementation tools.   */
/* ================================================== */
```

updown.attr

```
set_attribute find(port,"reset") "pad_location" -type string "P56"
set_attribute find(port,"clk") "pad_location" -type string "P72"
set_attribute find(port,"enable") "pad_location" -type string "P19"
set_attribute find(port,"up_dn") "pad_location" -type string "P20"
set_attribute find(port,"lsb_seg<0>") "pad_location" -type string "P49"
set_attribute find(port,"lsb_seg<1>") "pad_location" -type string "P48"
set_attribute find(port,"lsb_seg<2>") "pad_location" -type string "P47"
set_attribute find(port,"lsb_seg<3>") "pad_location" -type string "P46"
set_attribute find(port,"lsb_seg<4>") "pad_location" -type string "P45"
set_attribute find(port,"lsb_seg<5>") "pad_location" -type string "P50"
set_attribute find(port,"lsb_seg<6>") "pad_location" -type string "P51"
set_attribute find(port,"msb_seg<0>") "pad_location" -type string "P39"
set_attribute find(port,"msb_seg<1>") "pad_location" -type string "P38"
set_attribute find(port,"msb_seg<2>") "pad_location" -type string "P36"
set_attribute find(port,"msb_seg<3>") "pad_location" -type string "P35"
set_attribute find(port,"msb_seg<4>") "pad_location" -type string "P29"
set_attribute find(port,"msb_seg<5>") "pad_location" -type string "P40"
set_attribute find(port,"msb_seg<6>") "pad_location" -type string "P44"
```