Lab 7: From RTL to GDS

Objective

In this lab, we will learn how to transform a design from RTL to GDS

Reports

Please include any problems, findings and results of earch step in the report, especially:

1. Screenshot of design after each step
2. Generated scripts & reports

Introduction

To implement a design from RTL to GDS, there are many steps involved:

1. Simulate the design to verify its functionality
2. The RTL code should be synthesizable. we have to modify the code so that it can be synthesized by the synthesis tools.
3. Create a wrapper to fix the design generic and parameters as constants
4. Add the IO cells
5. Replace big lookup table with a ROM
6. Consider to replace the memory arrays as Register files or SRAM. SRAM can be single port or dual port
7. Set up the technology scripts and the tech files
8. Synthesis
9. Floorplan & IO Ring creation
10. Place & route
11. Formal verification
12. Parasitic Extraction & Static timing analysis
13. DRC/LVS
14. Metal fill and chip finish
15. Write out the GDSII

In this lab, we will use a VHDL source code from a generic FFT Design in Opencores: https://opencores.org/projects/versatile_fft.

The directory structures:

```
.
├── common
│   ├── common_setup.tcl
│   ├── common.tcl
│   └── warnings_to_ignore.tcl
├── env.sh
├── inputs
│   └── fft_engine_wrapper.sdc
├── sim
│   ├── Makefile
│   └── scripts
│       └── dump_waves.tcl
├── src
│   ├── rtl
│   │   ├── butterfly_d3.vhd
│   │   ├── dpram_inf.vhd
│   │   ├── dpram_rbw_inf.vhd
│   │   ├── fft_data_switch.vhd
│   │   ├── fft_engine.vhd
│   │   ├── fft_len.vhd
│   │   ├── fft_support.vhd
│   │   ├── icpx_mul_d3.vhd
│   │   ├── icpx_pkg.vhd
│   │   └── icpxram_rbw.vhd
│   └── tb
│       ├── fft_engine_tb.vhd
│       └── test_fft.m
└── versatile_fft
```

Laboratory tasks

Task 1. Simulate the design provided in this lab using VCS

One way to test the design is to generate the input and output data using a reference program (in Matlab for example). In this design, the authors use Matlab to generate the random data and then feed these random data into the VHDL design. After that, when the simulation finish, the data generated by the simulator will be compared with the reference results from the Matlab.

To generate the test data, you need to rn the test_fft.m in Matlab (or a Matlab replacement program). In this work, we use Octave so that we don't need to install matlab.

```
octave -q --eval 'addpath("../src/tb"); test_fft'
```

After running the above command, you should see that there are two new files are created:

```
data_in.txt
data_oct.txt
```

Write the simulation scripts using the following command:

1. Create the library folder and synopsys_sim.setup

```
mkdir -p work_lib
echo "WORK > work_lib" > synopsys_sim.setup
echo "work_lib: ./work_lib" >> synopsys_sim.setup
```

2. Compile the vhdl files from bottom-up

```
$ vhdlan -nc -kdb -work WORK ../src/rtl/fft_len.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/icpx_pkg.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/fft_support.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/dpram_rbw_inf.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/icpxram_rbw.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/butterfly_d3.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/icpx_mul_d3.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/dpram_inf.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/fft_data_switch.vhd
$ vhdlan -nc -kdb -work WORK ../src/rtl/fft_engine.vhd
$ vhdlan -nc -kdb -work WORK ../src/tb/fft_engine_tb.vhd
$ vcs -kdb -debug_access+all fft_engine_tb
```

3. Run the design and extract the waveform if necessary

```
./simv -ucli -do scripts/dump_waves.tcl
```

Question: Is the results from HDL design and the one from Matlab are exactly matched? Why?

Task 2. Modify the VHDL design to fix the design parameter

In this part, we will fix the parameters of the design (FFT_LOG_LEN to 4) and then modify the top-level design (fft_engine_wrapper) to use STD_LOGIC/STD_LOGIC_VECTOR only. This is because of the interface of the module are often in STD_LOGIC/STD_LOGIC_VECTOR.

1. Create a new vhdl file named fft_engine_wrapper.vhd under src/rtl with the contents as follows:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use ieee.math_complex.all;
library work;
use work.fft_len.all;
use work.icpx.all;
use work.fft_support_pkg.all;

entity fft_engine_wrapper is
  --generic (
  --  LOG2_FFT_LEN : integer := 4);       -- Defines order of FFT
  port (
    -- System interface
    rst_n     : in  std_logic;
    clk       : in  std_logic;
    -- Input memory interface
    din_re       : in  SIGNED(15 downto 0);        -- data input
    din_im       : in  SIGNED(15 downto 0);        -- data input
    valid     : out std_logic;
    saddr     : out unsigned(2 downto 0);
    saddr_rev : out unsigned(2 downto 0);
    sout0_re     : out SIGNED(15 downto 0);        -- spectrum output
    sout0_im     : out SIGNED(15 downto 0);        -- spectrum output
    sout1_re     : out SIGNED(15 downto 0);        -- spectrum output
    sout1_im     : out SIGNED(15 downto 0)        -- spectrum output
    );

end fft_engine_wrapper;

architecture structure of fft_engine_wrapper is
  constant LOG2_FFT_LEN : integer := 4;
  component fft_engine is
    generic (
      LOG2_FFT_LEN : integer);
    port (
      rst_n     : in  std_logic;
      clk       : in  std_logic;
      din       : in  icpx_number;
      valid     : out std_logic;
      saddr     : out unsigned(LOG2_FFT_LEN-2 downto 0);
      saddr_rev : out unsigned(LOG2_FFT_LEN-2 downto 0);
      sout0     : out icpx_number;
      sout1     : out icpx_number);
  end component fft_engine;
  signal din       : icpx_number;
  signal sout0     : icpx_number;
  signal sout1     : icpx_number;
begin  -- architecture structure


  fft_engine_1: entity work.fft_engine
    generic map (
      LOG2_FFT_LEN => LOG2_FFT_LEN)
    port map (
      rst_n     => rst_n,
      clk       => clk,
      din       => din,
      valid     => valid,
      saddr     => saddr,
      saddr_rev => saddr_rev,
      sout0     => sout0,
      sout1     => sout1);

  din.Re <= din_re;
  din.Im <= din_im;

  sout0_re <= sout0.Re;
  sout0_im <= sout0.Im;

  sout1_re <= sout1.Re;
  sout1_im <= sout1.Im;
end architecture structure;
```

2. Copy fft_engine_tb.vhd into fft_engine_wrapper_tb.vhd and modify the contents into the followings:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
--use ieee.math_complex.all;
library std;
use std.textio.all;
library work;
use work.fft_len.all;
use work.icpx.all;
use work.fft_support_pkg.all;

-------------------------------------------------------------------------------

entity fft_engine_wrapper_tb is

end fft_engine_wrapper_tb;

-------------------------------------------------------------------------------

architecture beh1 of fft_engine_wrapper_tb is

  type T_OUT_DATA is array (0 to FFT_LEN-1) of icpx_number;

  signal dptr                 : integer range 0 to 15;
  signal din, sout0, sout1    : icpx_number;
  signal saddr, saddr_rev     : unsigned(LOG2_FFT_LEN-2 downto 0);
  signal end_of_data, end_sim : boolean := false;
  component fft_engine_wrapper is
    port (
      rst_n     : in  std_logic;
      clk       : in  std_logic;
      din_re    : in  SIGNED(15 downto 0);
      din_im    : in  SIGNED(15 downto 0);
      valid     : out std_logic;
      saddr     : out unsigned(2 downto 0);
      saddr_rev : out unsigned(2 downto 0);
      sout0_re  : out SIGNED(15 downto 0);
      sout0_im  : out SIGNED(15 downto 0);
      sout1_re  : out SIGNED(15 downto 0);
      sout1_im  : out SIGNED(15 downto 0));
  end component fft_engine_wrapper;

  -- component ports
  signal rst_n : std_logic := '0';
  signal valid : std_logic;

  -- clock
  signal Clk : std_logic := '1';

begin  -- beh1
  fft_engine_wrapper_1: entity work.fft_engine_wrapper
    port map (
      rst_n     => rst_n,
      clk       => clk,
      din_re    => din.re,
      din_im    => din.im,
      valid     => valid,
      saddr     => saddr,
      saddr_rev => saddr_rev,
      sout0_re  => sout0.re,
      sout0_im  => sout0.im,
      sout1_re  => sout1.re,
      sout1_im  => sout1.im);

  Clk <= not Clk after 10 ns when end_sim = false else '0';

  -- waveform generation
  WaveGen_Proc : process
    file data_in         : text open read_mode is "data_in.txt";
    variable input_line  : line;
    file data_out        : text open write_mode is "data_out.txt";
    variable output_line : line;
    variable tre, tim    : real;
    constant sep         : string := " ";
    variable vout        : T_OUT_DATA;
  begin
    -- insert signal assignments here
    wait until Clk = '1';
    wait for 15 ns;
    wait until clk = '0';
    wait until clk = '1';
    rst_n <= '1';
    dptr  <= 0;
    l1 : while not end_sim loop
      if not endfile(data_in) then
        readline(data_in, input_line);
        read(input_line, tre);
       read(input_line, tim);
      else
        end_of_data <= true;
      end if;
      din <= cplx2icpx(complex'(tre, tim));
      if dptr < 15 then
        dptr <= dptr + 1;
      else
        dptr <= 0;
      end if;
      -- Copy the data produced by the core to the output buffer
      vout(to_integer(saddr_rev))       := sout0;
      vout(to_integer('1' & saddr_rev)) := sout1;
      -- If the full set of data is calculated, write the output buffer
      if saddr = FFT_LEN/2-1 then
        write(output_line, string'("FFT RESULT BEGIN"));
        writeline(data_out, output_line);
        for i in 0 to FFT_LEN-1 loop
          write(output_line, integer'image(to_integer(vout(i).re)));
          write(output_line, sep);
          write(output_line, integer'image(to_integer(vout(i).im)));
          writeline(data_out, output_line);
        end loop;  -- i
        write(output_line, string'("FFT RESULT END"));
        writeline(data_out, output_line);
        exit l1 when end_of_data;
      end if;
      wait until clk = '0';
      wait until clk = '1';
    end loop l1;
    end_sim <= true;

  end process WaveGen_Proc;
end beh1;
```

3. Simulate the new design and ensure that the results are the same

Task 3. Add the IO Cells to the chip

1. Take a look at the IO Cells documents at /home/dkits/synosys/SAED14nm/io_std/doc/SAED4nm_FinFET_Educational_Design_Kit.pdf
   List different kinds of IO Cells in the library and explain their characteristics.
   Which one should you choose to use as IO Cells to your chip? Explain why
2. Modify the fft_engine_wrapper.vhd to include the IO Cells, the signals will go through the IO Cells to the chip and also through the IO Cells to the external world.

An example of the IO mapping for fft_engine_wrapper.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use ieee.math_complex.all;
library work;
use work.fft_len.all;
use work.icpx.all;
use work.fft_support_pkg.all;

entity fft_engine_wrapper is
  --generic (
  --  LOG2_FFT_LEN : integer := 4);       -- Defines order of FFT
  port (
    -- System interface
    rst_n     : in  std_logic;
    clk       : in  std_logic;
    -- Input memory interface
    din_re    : in  signed(15 downto 0);  -- data input
    din_im    : in  signed(15 downto 0);  -- data input
    valid     : inout std_logic;
    saddr     : inout unsigned(2 downto 0);
    saddr_rev : inout unsigned(2 downto 0);
    sout0_re  : inout signed(15 downto 0);  -- spectrum output
    sout0_im  : inout signed(15 downto 0);  -- spectrum output
    sout1_re  : inout signed(15 downto 0);  -- spectrum output
    sout1_im  : inout signed(15 downto 0)   -- spectrum output
    );

end fft_engine_wrapper;

architecture structure of fft_engine_wrapper is
  constant LOG2_FFT_LEN : integer := 4;
  component fft_engine is
    generic (
      LOG2_FFT_LEN : integer);
    port (
      rst_n     : in  std_logic;
      clk       : in  std_logic;
      din       : in  icpx_number;
      valid     : out std_logic;
      saddr     : out unsigned(LOG2_FFT_LEN-2 downto 0);
      saddr_rev : out unsigned(LOG2_FFT_LEN-2 downto 0);
      sout0     : out icpx_number;
      sout1     : out icpx_number);
  end component fft_engine;
  component I1025_EW is
    port (
      DOUT  : out std_logic;
      PADIO : in  std_logic;
      R_EN  : in  std_logic);
  end component I1025_EW;
  component I1025_NS is
    port (
      DOUT  : out std_logic;
      PADIO : in  std_logic;
      R_EN  : in  std_logic);
  end component I1025_NS;
  component D8I1025_EW is
    port (
      DIN   : in  std_logic;
      PADIO : inout std_logic;
      EN    : in  std_logic);
  end component D8I1025_EW;
  component D8I1025_NS is
    port (
      DIN   : in  std_logic;
      PADIO : inout std_logic;
      EN    : in  std_logic);
  end component D8I1025_NS;

  signal din   : icpx_number;
  signal sout0 : icpx_number;
  signal sout1 : icpx_number;

  signal rst_n_from_pad   : std_logic := '0';
  signal clk_from_pad     : std_logic;
  signal din_re_from_pad  : signed(15 downto 0);
  signal din_im_from_pad  : signed(15 downto 0);
  signal valid_to_pad     : std_logic;
  signal saddr_to_pad     : unsigned(2 downto 0);
  signal saddr_rev_to_pad : unsigned(2 downto 0);
  signal sout0_re_to_pad  : signed(15 downto 0);
  signal sout0_im_to_pad  : signed(15 downto 0);
  signal sout1_im_to_pad  : signed(15 downto 0);
  signal sout1_re_to_pad  : signed(15 downto 0);
begin  -- architecture structure

  rst_n_io : I1025_NS
    port map (
      DOUT  => rst_n_from_pad,
      PADIO => rst_n,
      R_EN  => '1');

  clk_io : I1025_NS
    port map (
      DOUT  => clk_from_pad,
      PADIO => clk,
      R_EN  => '1');

  valid_io : D8I1025_NS
    port map (
      DIN   => valid_to_pad,
      PADIO => valid,
      EN    => '1');

  io2_gen : for i in 0 to 2 generate
    saddr_io : D8I1025_NS
      port map (
        DIN   => saddr_to_pad(i),
        PADIO => saddr(i),
        EN    => '1');
    saddr_rev_io : D8I1025_NS
      port map (
        DIN   => saddr_rev_to_pad(i),
        PADIO => saddr_rev(i),
        EN    => '1');
  end generate io2_gen;

  io_gen : for i in 0 to 15 generate
    din_re_io : I1025_EW
      port map (
        DOUT  => din_re_from_pad(i),
        PADIO => din_re(i),
        R_EN  => '1');
    din_im_io : I1025_EW
      port map (
        DOUT  => din_im_from_pad(i),
        PADIO => din_im(i),
        R_EN  => '1');
    sout0_re_io : D8I1025_EW
      port map (
        DIN   => sout0_re_to_pad(i),
        PADIO => sout0_re(i),
        EN    => '1');
    sout0_im_io : D8I1025_EW
      port map (
        DIN   => sout0_im_to_pad(i),
        PADIO => sout0_im(i),
        EN    => '1');
    sout1_re_io : D8I1025_NS
      port map (
        DIN   => sout1_re_to_pad(i),
        PADIO => sout1_re(i),
        EN    => '1');
    sout1_im_io : D8I1025_NS
      port map (
        DIN   => sout1_im_to_pad(i),
        PADIO => sout1_im(i),
        EN    => '1');
  end generate io_gen;

  fft_engine_1 : fft_engine
    generic map (
      LOG2_FFT_LEN => LOG2_FFT_LEN)
    port map (
      rst_n     => rst_n_from_pad,
      clk       => clk_from_pad,
      din       => din,
      valid     => valid_to_pad,
      saddr     => saddr_to_pad,
      saddr_rev => saddr_rev_to_pad,
      sout0     => sout0,
      sout1     => sout1);

  din.Re <= din_re_from_pad;
  din.Im <= din_im_from_pad;

  sout0_re_to_pad <= sout0.Re;
  sout0_im_to_pad <= sout0.Im;

  sout1_re_to_pad <= sout1.Re;
  sout1_im_to_pad <= sout1.Im;
end architecture structure;
```

3. Simulate the design to ensure that it function correctly.

Task 4. (Bonus) Replace the memory (array of flipflop) in the design with the SRAM macros

1. Take a look at the SRAM documents at /home/dkits/synopsys/SAED14nm/sram/doc/SAED14nm_FinFET_Educational_Design_Kit_Rev_1.1_15.07.2019.pdf
2. List different kinds of SRAM Cells in the library.
3. Create a wrapper for the SRAM module (for example, dpram_inf.vhd, dpram_rbw_inf.vhd)
4. Simulate the design with the newly created SRAM. Ensure that the design is still functioning.

Task 5. Synthesize the Design

1. Modify the scripts from the previous labs to include the IO Cells library
2. Write a timing constraint for the design
3. Synthesize the design with the IO Cells