EECS 627, Lab Assignment 2

1 Introduction

In this lab assignment, you will extend the process of designing your *trivial* chip. You will add two more blocks (a pseudo-random test pattern generator and a signature analyzer), constituting a BILBO to the design. The pseudo-random test pattern generator will be used to feed inputs to the multiplier and the signature analyzer will verify the correct functionality of the multiplier. Behavioral code for pattern generator(lfsr) and signature analyzer is provided and you will have to synthesize them to get structural verilog. You will do place-and-route (APR) all the multiplier, lfsr and signature analyzer separately. Finally all the blocks will be assembled at the top-level along with clock, power and ground routing. You will be using *Cadence Silicon Ensemble* for performing the block-level APR and top-level chip assembly.

While working through this lab, keep in mind the following: You should be looking at all of the files your tools are reading/writing on your behalf (most of which are plain text), especially in the beginning when you are still learning about how the tools work. You should read the manuals about each and every command you have in your scripts. Remember: **Do not blindly cut and paste from the samples that have been provided,** but instead use them as a guide for figuring out how the tools work.

2 Behavioral Verilog

The top-level design consists of a 4-stage pipelined multiplier, an 16-degree lfsr for generating pseudo-random inputs to the multiplier and a signature analyzer block for examining the output of the multiplier.

1. Pipelined Multiplier:

You will use the multiplier designed in lab 1, which has been verified against the golden brick and synthesized for maximum performance.

2. LFSR

A linear feedback shift register(lfsr) will be used for generating input vectors A and B for the pipelined multiplier. The lfsr implementing an *n*-degree polynomial goes through 2^n - 1 states before repeating the states. The only state not covered is the all-zero state. The behavioral code for a 16 bit lfsr has been provided on course webpage.

3. Signature Analyzer

A variant of lfsr can be used to compress the output of the pipelined multiplier. The behavioral code for the signature analyzer has been provided on the course webpage.

All the modules have be synthesized individually before place and route. The multiplier being used has already been synthesized in Lab1. You will have to synthesize the lfsr and signature analyzer modules to get a structural netlist before proceeding to the APR step of the design flow.

3 Floorplan and partitioning

Figure 1 gives a sample floorplan for the chip. The three blocks correspond to the three components instantiated at the top level. A rough guide for pin assignments is also annotated. This picture will guide us when writing our place and route scripts. Notice that we are planning to partition the design into three blocks, each of which will be synthesized and placed separately.

4 Place and route

The main tool for place and route is **sedsm** , ie. Silicon Ensemble. Make sure you have selected the latest version with swselect. While this tool has a graphical interface, it has a textual command driven interface. The graphical interface just presents forms for you to enter the needed information which is then printed out as com-

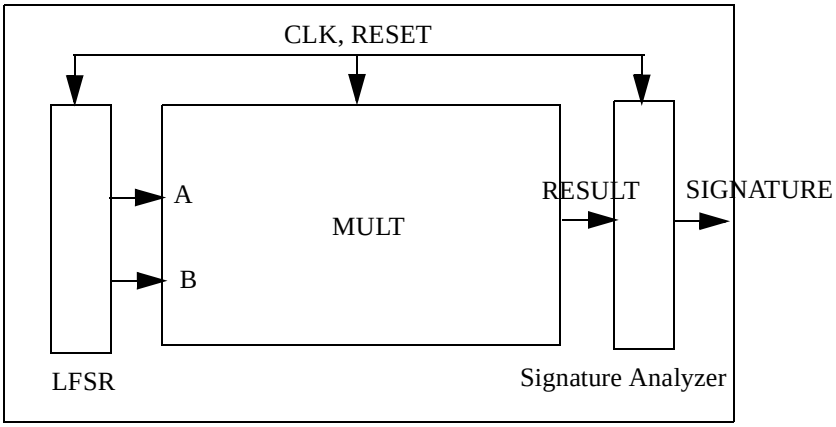

Figure 1. Sample floorplan of the design

mands. Thus by looking at the *se.jnl* log file the tool generates, you can quickly learn the commands you need to script the tool.

4.1 mult.se

I wrote a quick and dirty place and route script for the multiplier module. Most of this is generic and can be split out into common files to be shared across all of your modules. The things that are specific are related to the floorplan command, the pin placement, and the power stripes. For the LFSR module, I decided on an aspect ratio of 1 (ie. width/height) and a row utilization of 0.80. I decided this information by looking at the floorplan in Figure 1 as well as by trial and error (too high of a utilization causes the router to fail). The power stripes you need to decide based on the size and peak dissipation you expect from the module. I usually use the suffix.se for my silicon ensemble scripts, but the tool by default looks files with the suffix .mac. The script *lfsr.se* is as follows:

```
SET VARIABLE DRAW.ROW.AT ON;
FINPUT LEF FILENAME "/afs/engin.umich.edu/caen/generic/artisan/tsmc18/aci/sc/lef/tsmc18_6lm.lef" REPORTFILE "importlef.rpt" ;
SET VAR INPUT.VERILOG.POWER.NET "VDD";
SET VAR INPUT.VERILOG.GROUND.NET "VSS";
SET VAR INPUT.VERILOG.LOGIC.1.NET "VDD";
SET VAR INPUT.VERILOG.LOGIC.0.NET "VSS";
SET VAR INPUT.VERILOG.SPECIAL.NETS "VDD VSS";
INPUT CTLF INITFILE "./design.gcf" ;
INPUT VERILOG FILE "./mult.nl.v /afs/engin.umich.edu/caen/generic/artisan/tsmc18/aci/sc/verilog/tsmc18.v" LIB "cds_vbin" REFLIB "cds_vbin" DESIGN "cds_vbin.mult:hdl" ;

set v USERLEVEL EXPERT;
set v PLAN.REPORT.STAT " ";
FINIT FLOOR;
set v PLAN.REPORT.CALC " ";
# FINIT FLOORPLAN rowu 0.9 rowsp 0 blockhalo 3000 f x 4260000 abut xio 35000 yio 35000 ;
FINIT FLOORPLAN rowu 0.5 a 1 rowsp 0 blockhalo 3000 f abut ;
# FINIT FLOORPLAN rowu 0.9 rowsp 0 blockhalo 3000 f x 4260000 abut xio 35000 yio 35000 ;
FINIT FLOORPLAN rowu 0.5 a 1 rowsp 0 blockhalo 3000 f abut ;
set v UPDATECOREROW.BLOCKHALO.GLOBAL 4000;
window fit ;
REFRESH ;

IOPLACE AUTOMATIC STYLE EVEN TOPBOTTOMLAYER METAL4 RIGHTLEFTLAYER METAL3 ;
IOPLACE FileName template.ioc WRITE;
# now edit template.ioc using your floorplanning diagram and put the result in
# mult.ioplace
#IOPLACE FILENAME "./mult.ioc" STYLE EVEN TOPBOTTOMLAYER METAL2 RIGHTLEFTLAYER METAL3 ;
REFRESH ;
```

```
# define power stripes
BUILD CHANNEL;
CREATE RING NETS "VSS VDD" TYPE CORE_RINGS NO_EXCLUDE_SELECTED CENTER LAYER_TOP "METAL3" LAYER_BOTTOM "METAL3" LAYER_LEFT "METAL4" LAYER_RIGHT "METAL4" WIDTH_TOP "1720 " WIDTH_BOTTOM "1720 " WIDTH_LEFT "1720 " WIDTH_RIGHT "1720 " SPACING_TOP "560 " SPACING_BOTTOM "560 " SPACING_LEFT "560 " SPACING_RIGHT "560 ";

# If necessary, add power stripes
CREATE STRIPE NETS "VSS VDD" LAYER "METAL4" WIDTH "8000 " SET_TO_SET_DISTANCE 200000 SPACING "1800 " DIRECTION VERTICAL XLEFT_OFFSET 100000 XRIGHT_OFFSET 100000
#ADD STRIPE     NET "VSS" NET "VDD" DIRECTION Vertical LAYER METAL4 WIDTH 10000 SPACING 8000 COUNT 2 ALL ;
REFRESH;

# set variable for the placer
SET VAR QPLACE.TIMING.MODE "FALSE";
SET VAR TIMING.REPORTTIMING.LOGFILENAME "";
SET VAR QPLACE.PLACE.GROUTE.ANALYSIS "temp.congMap" ;
SET VAR QPLACE.PLACE.PIN "" ;
SET VAR QPLACE.CLOCK.BUFFER.SITE "8";
SET VAR QPLACE.PLACE.AREA "" ;
SET VAR QPLACE.REGION.RELAX.PCT -1 ;
SET VAR QPLACE.REGION.EXCLUSIVE FALSE ;
SET VAR QPWR.RSPF "" ;
SET VAR QPLACE.OPT.TIMING.TYPE "";
# run the placer
QPLACE NOCONFIG ;
# Save the design
SAVE DESIGN "PLACED" ;
FLOAD DESIGN "PLACED" ;

#DO the clock tree generation
OUTPUT LEF FILENAME PLACED.lef;
SET V OUTPUT.DEF.SPNET.WILDCARD TRUE ;
OUTPUT DEF FILENAME PLACED.def;
SET V OUTPUT.DEF.SPNET.WILDCARD FALSE ;
CTGEN FILENAME ctgen.cmd;
SET V ALLOW.EC TRUE;
INPUT DEF ECO FILENAME ctgen.def  NOKEEPDISTCELLS NOKEEPDISTNETS NOKEEPTIMING NOKEEPLEQ NOKEEPEEQ ;
SET V ALLOW.EC FALSE;
#route the clock
CLOCKROUTE ALL ;
REFRESH ;
SAVE DESIGN "CLOCKED" ;
FLOAD DESIGN "CLOCKED" ;
OUTPUT LEF FILENAME CLOCKED.lef;
SET V OUTPUT.DEF.SPNET.WILDCARD TRUE ;
OUTPUT DEF FILENAME CLOCKED.def;
SET V OUTPUT.DEF.SPNET.WILDCARD FALSE ;

#SET VAR SROUTE.FOLLOWPINS.FILL "TRUE";
#SROUTE FOLLOWPINS NET "VSS" NET "VDD" LAYER METAL1 WIDTH 1600 ;
#route power
CONNECT RING NET "VSS" NET "VDD" STRIPE BLOCK ALLPORT IOPAD ALLPORT IORING FOLLOWPIN ;
 REFRESH ;

 SET VAR WROUTE.SELECTNETS "" ;
 SET VAR WROUTE.ASSIGN.ROUTINGDIR "TRUE";
 SET VAR WROUTE.OPTIMIZATION "TRUE";
 SET VAR WROUTE.SELECTNETS "" ;
 SET VAR WROUTE.SELECTNET.FIRST FALSE ;
```

```
 SET VAR WROUTE.FINAL TRUE ;
 SET VAR WROUTE.GLOBAL TRUE ;
 SET VAR WROUTE.INCREMENTAL.FINAL FALSE ;
WROUTE NOCONFIG ;


 #add filler cells for spots there no standard cell was placed
 SROUTE ADDCELL MODEL FILL64 PREFIX FILL AREA  (-4000000 -4000000 ) (4000000 4000000 );
SROUTE ADDCELL MODEL FILL32 PREFIX FILL AREA  (-4000000 -4000000 ) (4000000 4000000 );
SROUTE ADDCELL MODEL FILL16 PREFIX FILL AREA  (-4000000 -4000000 ) (4000000 4000000 );
SROUTE ADDCELL MODEL FILL8 PREFIX FILL AREA  (-4000000 -4000000 ) (4000000 4000000 );
SROUTE ADDCELL MODEL FILL4 PREFIX FILL AREA  (-4000000 -4000000 ) (4000000 4000000 );
SROUTE ADDCELL MODEL FILL2 PREFIX FILL AREA  (-4000000 -4000000 ) (4000000 4000000 );
SROUTE ADDCELL MODEL FILL1 PREFIX FILL AREA  (-4000000 -4000000 ) (4000000 4000000 );

#Do an incremental route if something didn't work above
SET VAR WROUTE.FINAL TRUE;
SET VAR WROUTE.GLOBAL TRUE;
SET VAR WROUTE.INCREMENTAL.FINAL TRUE;
WROUTE NOCONFIG;
SAVE DESIGN "ROUTED";

OUTPUT DEF  FILENAME  "./mult.def" ;
OUTPUT LEF BLOCK  FILENAME  "mult.lef" ;
REPORT DELAY SDFOUTPUT FILENAME "./mult.APR.sdf" ;
OUTPUT  GDSII  MAPFILE  /usr/caen/generic/artisan/tsmc18/fb_tpz973g_230b/TSMCHOME/digital/silicon_ensemble/
tpz973g_230b/6lm/techfiles/opus.map  LIBNAME  CHIP  STRUCTURENAME  control  FILE  ./mult.gds2  REPORTFILE
LBRARY.gds2.jnl UNITS Thousands  ;
OUTPUT VERILOG FILE "mult_APR.v" ;
```

4.2 General Constraint File

To load the timing libraries, you need a general constraint file that specifies the process, voltage, and temperature, as well as pointing the tool at the appropriate timing info for the standard cell library. The file I use is as follows **design.gcf**:

```
(GCF
 (HEADER
  (VERSION  "GCF Version 1.2" )
  (DESIGN  "TYPICAL SAMPLE" )
  (DATE  "Thursday, April  8, 1999 - 10:34:29" )
  (PROGRAM  "BuildGates"  "v2.3"  "Cadence Design Systems, Inc." )
  (DELIMITERS  "/[]" )
  (TIME_SCALE 1E-9 )
  (CAP_SCALE 1000E-15 )
  (RES_SCALE 1E3 )
  (VOLTAGE_SCALE 1E0 )
 )
 (GLOBALS
  (GLOBALS_SUBSET ENVIRONMENT
   (PROCESS  1.0000 1.0000)
   (VOLTAGE  1.8000 1.8000)
   (TEMPERATURE  25.0000 25.0000)
   (OPERATING_CONDITIONS  "TYPICAL"  1.0000 1.8000 25.0000)
   (EXTENSION "CTLF_FILES"
    (/usr/caen/generic/artisan/tsmc18/aci/sc/tlf/typical.tlf )
   )
  )
 )
)
```

4.3 Pin Placement

In addition you will need to write by hand (potentially from the template file generated by the random pin-placer mode), a pin-placement constraint file. By looking at the floorplan diagram, you have to decide the pin placement such that the congestion is minimized during global routing. The sample template .ioc file (pin placement file format is .ioc for *silicon ensemble*) is as follows:

```
IOPLACEHEADER (
  (VERSION 5.3 )
  (DIVIDERCHAR "/" )
  (BUSBITCHARS "[]" )
)
TOP ( # IOs are ordered from left to right
  (IOPIN A[5] );
  (IOPIN A[4] );
  (IOPIN A[3] );
  (IOPIN A[2] );
  (IOPIN A[1] );
  (IOPIN A[0] );
  (IOPIN RESET );
  (IOPIN CLK );
  (IOPIN B[6] );
)
BOTTOM ( # IOs are ordered from left to right
  (IOPIN RESULT[15] );
  (IOPIN RESULT[14] );
  (IOPIN RESULT[13] );
  (IOPIN RESULT[12] );
  (IOPIN RESULT[11] );
  (IOPIN RESULT[10] );
  (IOPIN RESULT[9] );
  (IOPIN RESULT[8] );
  (IOPIN RESULT[7] );
)
LEFT ( # IOs are ordered from bottom to top
  (IOPIN B[5] );
  (IOPIN B[4] );
  (IOPIN B[3] );
  (IOPIN B[2] );
  (IOPIN B[1] );
  (IOPIN B[0] );
  (IOPIN A[7] );
  (IOPIN A[6] );
)
RIGHT ( # IOs are ordered from bottom to top
  (IOPIN RESULT[6] );
  (IOPIN RESULT[5] );
  (IOPIN RESULT[4] );
  (IOPIN RESULT[3] );
  (IOPIN RESULT[2] );
  (IOPIN RESULT[1] );
  (IOPIN RESULT[0] );
  (IOPIN B[7] );
)
IGNORE ( # IOs are ignored(not placed) by IO Placer
)
```

4.4 Running sedsm

I usually run sedsm in graphical mode, so I can visually inspect the results when things go wrong. (and things will go wrong) Since the program is ill-behaved and tends to generate hundreds of useless temporary files, it is safest to run the thing from a subdirectory in */usr/tmp*. Also, the speed of execution is 2-3X faster when executing with a local disk as the work directory as opposed to AFS.

```
mkdir /usr/tmp/my_sedsm
cd /usr/tmp/my_sedsm
sedsm -m=500 &
```

The -m option is for allocating memory to the process. When the program starts you get an empty window. Go to File – Execute and type in the full pathname of the script **mult.se** . Don't forget to edit this file to change the pathnames to your own files. You may also have to change the aspect ratio, the size of your floorplan, the number of power stripes etc in order to get a compact layout. Its a bit annoying, but you should put the full directory names and paths of your files in your scripts so that you can run the tool from a different directory than your source code, to avoid confusion. After a bit of running (during which the graphical display usually freezes, you should see your lfsr module placed and routed. Most likely you have to hit View–Redraw to see anything. Also, more than likely, something has gone horribly wrong with your scripts. The best place to debug this is from the *se.jnl* file that will be made in the working directory. This is a record of each command executed along with the results of execution (in comments). One typical mistake is to forget the trailing **;** after each command. Once you gain experience with the many limitations and quirks of the tool things will go much more smoothly. Typically I restart the tool periodically and remove the contents of the work directory to ensure correct operation. You should remove the unnecessary files and directories before invoking sedsm again because these files may cause the tool to crash.

Once everything is working you should get several output files, the main files of interest are **mult.APR.sdf** and **mult_APR.v**. **mult.APR.sdf** is an extracted timing file that lets you backannotate your verilog simulations. Most of you would have noticed during synthesis in Lab1 that the sdf file generated by design compiler has no information about the wire delay and all the delay values corresponding to the wires is zero. **mult_APR.v** is the verilog description of the design after the placement. Note that your placed design will be different from the synthesized structural netlist because of the insertion of clock tree during APR and **mult_APR.v** will be used during LVS of the final layout in Lab3. Hence it is important to verify the functionality of the verilog file generated by *silicon ensemble* with proper back-annotation of the corresponding sdf file.

4.5 Clock Tree Generation

Silicon Ensemble has an inbuilt tool called **ctgen** for generating clock tree for specified constraints. You will need two files: **ctgen.cmd** and **ctgen.const** for invoking the **ctgen** tool. A sample **ctgen.cmd** containing the environment setup commands is given below:

```
set_rundir 'CTGEN'
set_format se
read_technology
 lef_files '/afs/engin.umich.edu/caen/generic/artisan/tsmc18/aci/sc/lef/tsmc18_6lm.lef'
 gcf_env_file 'design.gcf'
read_design
  def_file 'PLACED.def'
read_constraints
  constraints_file 'ctgen.const'
build_clocktrees
remove_overlaps
write_design
 def_file 'ctgen.def'
```

ctgen.cont specifies the constraints on the clock skew, delay and transition time of the clock signal. A sample **ctgen.cmd** file is given below:

```
specify_tree
root_iopin 'CLK'
set_constraints
waveform 0.01 2.49 0.01 2.49
min_delay 0
max_delay 0.4
max_skew .04
```

max_transition .01

After running the clock tree generation tool, you can look at the output in se.jnl file which should be as follows:

```
#---{clockRoute : ========================================}
#---{clockRoute : ============ ClockRoute Result ============}
#---{clockRoute : ========================================}
#---{clockRoute : Tree 1: Clock tree root: PIN CLK}
#---{clockRoute :        M1 wirelength: 199 [4%] -> 199 [4%] (100%).}
#---{clockRoute :        M2 wirelength: 1971 [40%] -> 1971 [40%] (100%).}
#---{clockRoute :        M3 wirelength: 2657 [54%] -> 2657 [54%] (100%).}
#---{clockRoute :        M4 wirelength: 67 [1%] -> 67 [1%] (100%).}
#---{clockRoute :   Wirelength: 4894 -> 4894 (100%).}
#---{clockRoute :   Min delay :348 ps -> 348 ps (100%)}
#---{clockRoute :   Max delay :376 ps -> 376 ps (100%)}
#---{clockRoute :   Max skew  :28 ps -> 28 ps (100%)}
```

4.6 Using Silicon Ensemble for Block-Level APR and Global Routing

There are slight differences in .se files for the block-level APR and global APR. The block level APR can be automated and you can execute your .se script in one go. In global routing, the lef files containing the metal routing information for different instantiated modules (which have been placed using block level APR) need to be read in, along with the verilog netlists. You have to change the line 'CLASS MACRO" inside all the lef files to "CLASS BLOCK" before using them in global routing. After reading the files, you will find all the instantiated blocks at the bottom-left corner in your *silicon ensemble* GUI. These blocks have to be placed by hand inside the chip area before initiating the pin placement, power and signal routing. You will use the "move" command in the GUI for this purpose.

5 Deliverables

After working through this tutorial you should have the following items to submit to your GSI in an organized directory tree, which you should submit as a tar file, Detailed directions on submitting lab assignments will be posted on the course web page. **Do not include any other file in your tar ball**. Your submitted directory should have *only* the following files:

– README.txt: text file containing a brief description of any problems you encountered and how you solved them.

– detailed floorplan

– synthesized structural verilog for mult, lfsr and signature modules

– place and route (.se) scripts for mult, lfsr, signature blocks and global routing.

– pin placement (.ioc) files for mult, lfsr, signature and top level.

– ctgen.cmd and design.gcf file.

– ctgen.const files for mult, lfsr and signature blocks.

– se.jnl for the three block-level APRs and the global APR