NC State University
ECE Department ECE 720 W. Rhett Davis

Homework #2 Solution

This solution begins with the Homework #1 solution. We start by modifying the Makefile to run each iteration of the flow and capture the results, as given below:

Makefile

```
UTIL=0.5

pnr: synth
	cd icc2rm && $(MAKE) clean
	$(MAKE) -C icc2rm
	python parse_reports.py $(UTIL)

synth:
	python set_constraints.py $(UTIL)
	$(MAKE) -C dcrm

setup:
	echo date,target_util,route_auto_util,route_auto_len,route
_auto_viol,route_auto_time > results.csv

clean:
	cd dcrm && $(MAKE) clean
	cd icc2rm && $(MAKE) clean
```

Note that the Makefile uses the variable UTIL to set the target utilization on each step. The python script set_constraints.py is called to set the target utilization before running the DC-RM flow, and the parse_reports.py script is called after the ICC2-RM to capture the results in the CSV file. The “setup” target is used to initialize the CSV file. The pnr target is also changed to clean the directory every time it runs, which is a simple way to ensure that it re-runs the ICC2-RM flow every time we execute “make”.

Next we move to the dcrm directory, which we set up much like it was in the Homework #1 solution, with a very similar Makefile. The scripts are modified as recommended in class to build the xbar design instead of the counter design. These changes are straight-forward and not re-created here. The important thing to remember is to change the MAX_ROUTING_LAYER in dcrm/rm_setup/common_setup.tcl from MINT5 to MINT3, as required in the problem statement.

Next, let’s look at the set_constraints.py:

set_constraints.py

```
import re, sys

src=open('dcrm/rm_dc_scripts/dc.tcl.template')
dest=open('dcrm/rm_dc_scripts/dc.tcl','w')
for line in src:
  m=re.search(r'^set_utilization',line)
  if m:
    dest.write('set_utilization '+sys.argv[1]+'\n')
  else:
    dest.write(line)
src.close()
dest.close()
```

This script goes through a template script (dc.tcl.template) line-by-line and copies it into the dc.tcl file. When it encounters the "set_utilization" command, it changes the target utilization using the first argument sent to the script. The important thing to remember here is that we must also copy the file *dcrm/rm_dc_scripts/dc.tcl* to *dcrm/rm_dc_scripts/dc.tcl.template*, and that the **set_utilization** command must be added to the template before running the flow.

Next, we'll move to the icc2rm directory, which is also set up much the same as it was in the Homework #1 solution. There are a few changes:

- The file icc2rm/Makefile is set to run the route_auto target, instead of route_opt
- The file icc2rm/rm_setup/icc2_common_setup.tcl is modified for the xbar design instead of the counter design, as described in the lecture
- The file icc2rm/rm_setup/Makefile_pnr is modified to create a date-file called route_auto.begin (similar to the way the synth.begin file was created in the Homework #1 solution) Here's a diff on my Makefile_pnr:

```
$ git diff Makefile_pnr
diff --git a/rm_setup/Makefile_pnr b/rm_setup/Makefile_pnr
index eae82c4..a130094 100755
--- a/rm_setup/Makefile_pnr
+++ b/rm_setup/Makefile_pnr
@@ -35,6 +35,7 @@ clock_opt_opto: setup clock_opt_cts
        $(ICC2_EXEC) $(OPTIONS) -f ./rm_icc2_pnr_scripts/clock_opt_opto.tcl | te

 route_auto: setup clock_opt_opto
+       date > route_auto.begin
        $(ICC2_EXEC) $(OPTIONS) -f ./rm_icc2_pnr_scripts/route_auto.tcl | tee -i

 route_opt: setup route_auto
```

Next, let's look at the parse_results.py script, which is the most complex part of this flow:

parse_reports.py

```
import re, os, sys

target_util=sys.argv[1]

t1=os.path.getmtime('icc2rm/route_auto.begin')
t2=os.path.getmtime('icc2rm/route_auto')
route_auto_time=str(t2-t1)


f=open('icc2rm/rpts_icc2/route_auto.check_routes')
for line in f:
  m=re.search(r'TOTAL VIOLATIONS =\s+(\d+)',line)
  if m:
    route_auto_viol=m.group(1)
    continue
  m=re.search(r'Total Routed Wire Length =\s+(\d+)',line)
  if m:
    route_auto_len=m.group(1)
    break
f.close()

f=open('icc2rm/rpts_icc2/route_auto.report_utilization')
for line in f:
  m=re.search(r'Utilization Ratio:\s+([0-9\.]+)',line)
  if m:
    route_auto_util=m.group(1)
    break
f.close()

f=open('icc2rm/route_auto')
route_auto_date=f.readline().strip()

f=open('results.csv','a')
f.write(route_auto_date+',')
f.write(target_util+',')
f.write(route_auto_util+',')
f.write(route_auto_len+',')
f.write(route_auto_viol+',')
f.write(route_auto_time+'\n')
f.close()
```

This script gathers the required results from each report file and appends a line to the CSV file

Finally, we need setup scripts, as with Homework #1 (although in this case, the varsetup.sh script is not necessary, since these python scripts run equally well with the default Python 2):

addsetup.sh

```
add synopsys2019
```

varsetup.sh

```
source /afs/eos.ncsu.edu/lockers/research/ece/wdavis/tools/anaconda3/setup.sh
```

Once all of these files are created, the solution to the homework is generated with the following steps:

```
$ source addsetup.sh
$ source varsetup.sh
$ make setup
$ make
$ make UTIL=0.9
$ make UTIL=0.7
$ make UTIL=0.8
$ make UTIL=0.75
$ make UTIL=0.73
$ make UTIL=0.72
```

The first execution of "make" initialized the CSV file, and the second runs the flow with the default target utilization of 0.5. Each additional run adjusts the target utilization as needed to search for a utilization that gets a clean route with no violations. The resulting CSV file is shown below:

results.csv

```
date,target_util,route_auto_util,route_auto_len,route_auto_viol,route_auto_time
Mon Sep 16 10:14:04 2019,0.9,0.8430,55131,480,591.461017132
Mon Sep 16 10:30:33 2019,0.7,0.6551,61106,0,61.5541050434
Mon Sep 16 10:44:00 2019,0.8,0.7443,59187,22,415.279694557
Mon Sep 16 10:50:39 2019,0.75,0.6986,59776,2,104.910173655
Mon Sep 16 11:00:51 2019,0.73,0.6786,61494,4,100.213164568
Mon Sep 16 11:25:34 2019,0.72,0.6747,60728,0,68.960111618
```

As you can see, I followed a binary search until arriving at two utilizations: 0.73 (which has 4 violations) and 0.72 (which has no violations).