Synthesis of a Million Gates

Don Mills

Lockheed Martin Corp.

ABSTRACT

The challenge of attacking a million gate design project requires many tools and approaches. This paper describes an approach used to manage the design flow and synthesis of such a design. Much of the design flow is automated, and the automation approach with its scripts is presented herein. Using this approach, a full synthesis of the design incorporates over 250 synthesis runs and takes approximately 40 hours to run from start to finish using only 8 compiler licenses. The frequencies for the clocks in the design range from 1.5 MHz to 180 MHz.

Introduction

There are many approaches to applying Synopsys scripts and tools to the back end processing of an ASIC. The approaches presented in this paper in no way intend to be the best or fastest, but are intended to provide a working approach that addresses many of the problems and difficulties of managing a very large design through the synthesis tools.

In order to reduce the magnitude and complexity of synthesizing the design, as much automation as possible was instituted in the process. This automation began with utilizing a file management tool. Then a default synthesis compile script was developed. When the default compile script was not sufficient, an additional custom script was utilized. In most cases, the custom script only needed to add one or two additional constraints to those defined in the default script. The synthesis compiling was managed by use of gmake. gmake was chosen because of its capability of building the makefile targets in parallel. The makefile uses a home grown network script to determine which workstation is the best choice before rlogging onto a workstation and building a target. At the completion of synthesis, a script is used to summarize all the report files into a condensed simple table. This table provided a quick reference to where the trouble spots were in the design.

Basic Coding Guidelines

In order to facilitate the scripts and allow the maximum amount of automation, a few basic coding guidelines were used. When the guidelines were adhered to, the default scripts for synthesis worked great. Often, custom scripts were required to augment the default script when these guidelines were not or could not be followed. Most of these guidelines have been recommended by Synopsys for years. They are:

1. register all the outputs of logic blocks;
2. only one functional clock per logic block;
3. no mixing of logic and hierarchy blocks, the exception to this is memories;
4. no gating the reset even though the reset was synchronous.

By registering the outputs, the complexity of the synthesis scripts was reduced enormously. The scripts could then take advantage of almost the whole clock period for logic within the block. If the design still had timing problems, custom scripts and a characterize-compile script were used to fix the timing. Also recoding the design and register balancing were used to fix timing. Register balancing was one of the big life savers for this project.

Restricting the logic blocks to one functional clock sped up synthesis and allowed the default script to constrain all the IO for the block for the single clock. Exceptions to this rule required custom scripts to override the default script's constraints of the IO and clock definitions.

Mixing logic cells with hierarchy is synthesizable but adds much to the complexity of the synthesis compile scripts, and causes longer synthesis run times. Additionally, depending on the vendor, this approach can be difficult to layout.

Even though the reset for the project was a synchronous reset, restrictions were placed on any gate manipulation of it. Because of the high fanout of the reset line, special approaches (similar to those used for clocks) were used to buffer reset. When the reset was gated with other control logic, these special buffering approaches broke down. Having control of the entire reset net from the top level of the design, layout considerations were able to be taken into account as part of the buffering scheme. This was very important for this project because of the extremely high fanout of the reset.

File Management

For various reasons at the beginning of the project, there were no file management approaches instituted. But as the project grew, it became obvious that some sort of management was needed. By the time file management was instituted, about 95% of the files for the project were coded. It was at this time that the efforts for the top level simulations started to ramp up and the resulting code changes to the files became unmanageable. After doing some research on the web, CVS was chosen to be the file management tool for the project. Even at this point in the project there was no schedule slip for instituting CVS, even though most files were already created. Once it was determined how to utilize CVS, it only took an afternoon to apply CVS to the project. By this time, the project consisted of more than a thousand files to be maintained. CVS was picked because it would maintain directory structures, as well as files, and because it allowed concurrent editing of files. Many engineers who were unfamiliar with the concept of concurrent editing of files were very skeptical and cautious of the tool and approach. But after a very short period of using this tool, they decided that there was nothing to be concerned with. The project team had more than 15 designers simulating and modifying code at the same time others were fixing unsynthesizable portions of code. Not very often can a project start using a new tool in the middle of the design cycle and experience no negative repercussions as a result.

There are a number of web sites that provide information about file management tools. Some the best starting places are:

Configuration Management Yellow Pages:
www.cs.colorado.edu/users/andre/configuration_management.html

Configuration Management Frequently Asked Questions:
www.iac.honeywell.com/Pub/Tech/CM/CMFAQ.html

Automation of the Synthesis

For a design of this size and magnitude, automation was a must for two basic reasons. First, there were too many designs to be synthesized to allow for a script to be written for each design. At last count, the complete synthesis of the project incorporated around 260 synthesis runs. The second reason for applying automation was to have the synthesis runs completed in a reasonable

amount of time. At the start of the project, the division only had 4 synthesis licenses. These were being used by three big projects and about 12 designers. As the three projects started hitting synthesis hard, it soon became obvious that 4 synthesis compiler licenses would be very inadequate. Adjustments were made by getting separate FPGA licenses and an additional 4 compiler licenses.

There are many aspects to the process flow that supported an automation approach. The key tools that made automation feasible were gmake and some network scripts that selected the best available workstation on the network. Other supporting tools were scripts that determined if there were licenses available, and the synthesis default and custom scripts that interfaced with the makefile. When the makefile would build a target, it would first wait until a license was available, then rlog onto the current best workstation. Next, a number of parameters were set using environmental variables, and then synthesis was called. The environmental variable parameters were used to configure the synthesis default scripts. This was how one script could be used to successfully synthesize 80-90% of the design.

```
/*************************************************************
syn_def_comp.scr
                                        copyright 1996
                                        Lockheed Martin WBS
                                        All Rights Reserved


   File Name : syn_default.scr
   Author      : Don Mills
   phone       : x3270
   Title       : Synopsys default script

   Purpose   : this file is used as the default script for
                  first pass synthesis

*************************************************************
Modification History:

   Rev #  Date      Author  Description of Change
         (yy-mm-dd)
   1.00   96-02-13  drm    create
   1.01   96-03-19  drm     added if statements to ungroup
                     and/or uniquify

*************************************************************/

sh date
sh hostname

SYN_LIB_NAME      = get_unix_variable("SYN_LIB_NAME")
ENTITY_NAME       = get_unix_variable("ENTITY_NAME")
CLK_NAME          = get_unix_variable("CLK_NAME")
CLK_PERIOD        = get_unix_variable("CLK_PERIOD")
RESET_NAME        = get_unix_variable("RESET_NAME")
MAP_EFFORT        = get_unix_variable("MAP_EFFORT")
SYN_UNIQUIFY      = get_unix_variable("SYN_UNIQUIFY")
SYN_UNGROUP       = get_unix_variable("SYN_UNGROUP")
SYN_PARAMS        = get_unix_variable("SYN_PARAMS")
SYN_V_PART        = get_unix_variable("SYNOPSYS_V_PART")
CUSTOM_SCR        = get_unix_variable("CUSTOM_SCR")

if (SYN_PARAMS == "none") {
    elaborate -lib SYN_LIB_NAME ENTITY_NAME
} else {
    elaborate -lib SYN_LIB_NAME ENTITY_NAME -parameters SYN_PARAMS
}
remove_license VHDL-Compiler
write -h -o ENTITY_NAME + "_unsynth.db"

create_clock -period CLK_PERIOD CLK_NAME

if (SYN_UNIQUIFY == 1) {
```

```
   uniquify
}

if (SYN_UNGROUP == 1) {
   ungroup -all -simple_names -flatten
}

set_operating_conditions WC8

set_driving_cell -cell dfntsq1 -pin q all_inputs()
set_drive 0 CLK_NAME

set_load load_of (SYN_V_PART + "/nd02d1/a1") all_outputs()

set_input_delay 1.25 -max -clock CLK_NAME all_inputs()
set_input_delay 0.0  -max -clock CLK_NAME CLK_NAME

if (RESET_NAME != 0) {
   set_input_delay 3.0 -max -clock CLK_NAME RESET_NAME
   set_drive 0 RESET_NAME
}

set_output_delay 3 -max -clock CLK_NAME all_outputs()

set_dont_touch_network CLK_NAME

set_max_transition 1 find(design,"*") /* over-ride vendor lib */

if (CUSTOM_SCR == 1) {
   include "/user/om73/make_control/syn_custom/"  + \
            ENTITY_NAME + ".scr"
}


compile -map_effort MAP_EFFORT


check_design
check_timing

write -h -o "." + current_instance + ".db"

report_area
report_timing -path full -delay max -max_paths 5 -nworst 1 \
              -to all_registers(-data_pins) + all_outputs()
report_clock

sh date

exit
```

FIGURE 1 - The Default Synthesis Script

The default script begins by reading the environmental variables and converting them to local Synopsys variables. If the design being read needs to have a generic value set at elaboration, then the generic value is set using the -parameters option. After elaboration, the VHDL-compiler license is released. It will be re-obtained during the compiling process if needed. If it is not needed, the license is made available for other designers to use. The design is then written out to allow future access to the elaborated but unsynthesized design, without the need to perform the elaboration and tie up a VHDL-compiler license again. This file is a very useful starting point when developing a custom synthesis compile script.

With the design elaborated and in memory, the default script then prepared the design for compilation by defining the operating conditions and a default version of the design's external environment. Variables are also checked to determine if the design should be uniquified and/or ungrouped. Special consideration is given to the clock and reset when defining the input constraints. The final variable checked before compiling the design is the CUSTOM_SCRIPT variable. This allows for the inclusion of a custom script to add any additional constraints or override any constraints previously set. Examples and discussion of the custom script will follow later.

The design was then compiled and the resulting design was written out. Finally, a number of reports were generated. The results of the report commands were stored in the .out file as directed by the makefile command.

Because of the number of synthesis scripts run for this project, it became very impractical to review each of the .out files individually. Therefore, an awk file was generated to extract information from all the .out files.

```
=========================================================================
File:         BistTop.out
Run: guitar   from [10:22 on Dec 9, 1996]  to [10:24 on Dec 9, 1996]
ENTITY_NAME   BistTop
-------------+--------+--------------+------------+--------------------
CLK_PERIOD    | 40     | Com_Area     |    1088.67 | SLACK(Clk[40])
MAP_EFFORT    | medium | NonCom_Area  |     297.33 +--------------------
SYN_UNIQUIFY  | NO     | Sum          |    1386.00 | (MET)         27.47
SYN_UNGROUP   | NO     | Area (sq mils)|     600.60 | (MET)         28.04
              |        | Length (mils) |      24.51 | (MET)         28.46
              |        |               |            | (MET)         28.72
              |        |               |            | (MET)         29.39
=========================================================================
File:         CONV_ENC.out
Run: piano    from [15:46 on Dec 16, 1996]  to [16:26 on Dec 16, 1996]
ENTITY_NAME   CONV_ENC
-------------+--------+--------------+------------+--------------------
CLK_PERIOD    | 12     | Com_Area     |    1168.33 | SLACK(CP_IN[12])
MAP_EFFORT    | medium | NonCom_Area  |     813.00 +--------------------
SYN_UNIQUIFY  | NO     | Sum          |    1981.33 | (MET)          1.48
SYN_UNGROUP   | YES    | Area (sq mils)|     858.58 | (MET)          2.10
              |        | Length (mils) |      29.30 | (MET)          2.10
              |        |               |            | (MET)          2.11
              |        |               |            | (MET)          2.11
=========================================================================
File:         Cordic10.out
Run: violin   from [08:18 on Dec 9, 1996]  to [08:21 on Dec 9, 1996]
ENTITY_NAME   Cordic10
-------------+--------+--------------+------------+--------------------
CLK_PERIOD    | 5      | Com_Area     |    4644.00 | SLACK(Clk[5])
MAP_EFFORT    | medium | NonCom_Area  |    4907.33 +--------------------
SYN_UNIQUIFY  | NO     | Sum          |    9551.33 | (VIOLATED)    -1.07
SYN_UNGROUP   | NO     | Area (sq mils)|    4138.91 | (VIOLATED)    -1.07
              |        | Length (mils) |      64.33 | (VIOLATED)    -1.07
              |        |               |            | (VIOLATED)    -1.07
              |        |               |            | (VIOLATED)    -1.04
=========================================================================
File:         Cordic12.out
Run: lynx     from [09:40 on Dec 7, 1996]  to [09:45 on Dec 7, 1996]
ENTITY_NAME   Cordic12
-------------+--------+--------------+------------+--------------------
CLK_PERIOD    | 5      | Com_Area     |    6912.33 | SLACK(Clk[5])
MAP_EFFORT    | medium | NonCom_Area  |    8412.99 +--------------------
SYN_UNIQUIFY  | NO     | Sum          |   15325.32 | (VIOLATED)    -1.40
SYN_UNGROUP   | NO     | Area (sq mils)|    6640.97 | (VIOLATED)    -1.40
              |        | Length (mils) |      81.49 | (VIOLATED)    -1.40
              |        |               |            | (VIOLATED)    -1.40
              |        |               |            | (VIOLATED)    -1.40
=========================================================================
```

FIGURE 2 - Excerpt From Summary.out File

The summary from each .out file tells which workstation the synthesis job was run on, and the start and finish times of the job. The variables that were most likely to be utilized to help the design meet timing are included in the summary table. Finally, a summary of the area and timing is included. The area in the table only includes the combinational and non-combinational area. The sum and total area were calculated by formulas given by the ASIC vendor.

When a design is shown to not meet timing, there are a variety of approaches that could be taken to resolve the issue. The choice of approach depends on many factors and is not the subject of this paper. When constraints needed to be modified or added from those set in the default compile script, a custom script was used. Out of the 260 Synopsys runs for this project, only about 50 custom scripts were required. More than half of these required only a few lines of additional constraints, such as those that follow:

```
      set_false_path -to MuxTestPoints*[*]
      set_false_path -from uPAddr_Bus[*]
      set_false_path -from Bist_En
      set_false_path -from MuxEnc_Sel

      set_multicycle_path 2 -to U3/U1/FF*/DOUT_reg[*]
      set_multicycle_path 2 -from U1/U2/MappingAdr_R_reg[*] -to
U1/U6/WrAddr_R_reg[*]
```

In a few cases, significant changes and additions were required to be made to the environment setup by the default script. A custom script was used to make the changes, run special compilation approaches, generate the reports, and exit the process. An example of such a custom script is included below. This script was used to setup the mux hooks built into Synopsys version 3.4. In order for the mux hooks to work, some attributes had to be set prior to elaboration.

```
/*****************************************************************************
  Custom script to infer a 64 bit mux
*/
sh date

remove_design -designs

hdlin_infer_mux      = "all"
hdlin_mux_size_limit = 64

elaborate -lib SYN_LIB_NAME ENTITY_NAME

remove_license VHDL-Compiler
write -h -o ENTITY_NAME + "_unsynth.db"

create_clock -period CLK_PERIOD CLK_NAME

uniquify
```

```
compile -map_effort MAP_EFFORT

set_operating_conditions -library "vsc783" "WC8"
compile  -map_effort high

check_design
check_timing

write -h -o "." + current_instance + ".db"

report_area
report_timing -path full -delay max -max_paths 5 -nworst 1 -to all_registers(-
data_pins) + cd all_outputs()
report_clock

sh date
exit
/*****************************************************************************/
```

FIGURE 3 - Custom Script for Mux Inference

It should be noted that this script contains two compile commands. It was found that if all the design constraints were placed on the design as part of the first compile, the design would not meet timing. However, if only the clock was defined for the first pass compile and the remaining constraints added as part of the second compile, the design met timing. This might be fixed in 3.5?

The Makefile

Makefiles and gmake were used to control the processing of the design. There were three significant tasks performed or managed by the makefile. The first task was to make sure a Synopsys compile license was available. If one was not available, then the makefile waited until a license was available before proceeding. The second task was to determine the best workstation on the network and rlog onto that workstation, setup environment settings, enter dc_shell, and execute the synthesis script. Finally, there were special hooks built in to and managed by the makefile to allow the makefile targets to be built in parallel. gmake was chosen because of its ability to build targets in parallel.

```
#########################################################################
#                                       copyright 1996
#                                       Lockheed Martin WBS
#                                       All Rights Reserved
#
#   File Name   : Makefile.synth
#   Author      : Don Mills
#   phone       : x3270
#   Title       : Makefile for project: (BEM)
#
```

```
#   Purpose     : manage the synthesis of the project
#
#   Usage       :  gmake -j 8 -f Makfile.syn
#
#########################################################################
#   Modification History:
#
#   Rev #  Date      Author   Description of Change
#         (yy-mm-dd)
#   1.00   96-02-12   drm     created
#
#########################################################################
#
#   General definitions
#
SHELL=          /bin/csh

VLSI_VERSION=    7
SYNOPSYS_V_PAD=  vsc${VLSI_VERSION}p31
SYNOPSYS_V_PART= vsc${VLSI_VERSION}83

WORKAREA=   ${PROJECT}
MAKE_SCR_DIR=     ${WORKAREA}/make_control


#SYNTH_NODE=     remsh `wsavail_mine`
#SYNTH_NODE=     remsh violin
SYNTH_NODE=     remsh `wsavail_mine -735 -s20`

SYN_DIR=          ${WORKAREA}/synopsys_${SYNOPSYS_VER}
SYN_LIB_DIR=      ${SYN_DIR}/OM73_lib
SYN_LIB_NAME=   OM73_lib
SYN_SCR_DIR=      ${SYN_DIR}/scr
SYN_DB_DIR= ${SYN_DIR}/${SYNOPSYS_V_PART}_dbs
SYN_CUST_DIR=   $(MAKE_SCR_DIR)/syn_custom
SYN_mSCR_DIR=     ${MAKE_SCR_DIR}/syn_default

SYN_V_VER=  ${SYNOPSYS_V_VER}

#WAIT_FOR_DC=
WAIT_FOR_DC=    /user/synopsys/bin/wait4dc_bat_low.csh
#WAIT_FOR_DC=   /user/synopsys/bin/wait4dc_bat_low1.csh
#WAIT_FOR_DC=   /user/synopsys/bin/wait4dc_batch.csh
#WAIT_FOR_DC=   /user/synopsys/bin/wait4dc.csh

DC_SETUP=    hostname; setupbem; cd ${SYN_DB_DIR}; \
             /user/synopsys/bin/wait4dc.csh
#DC_SETUP=  hostname; setupbem; cd ${SYN_DB_DIR}

NICE_IT=

VHDL_COMP_NODE= remsh `wsavail -sun`

#########################################################################
```

```
real_all_syn :    ${MAKE_SCR_DIR}/SLEEP_SOME_1   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_2   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_3   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_4   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_5   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_6   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_7   \
            ${SYN_DB_DIR}/Rcv_Core.db      \
           ${SYN_DB_DIR}/Xmt_Core.db      \
           ${SYN_DB_DIR}/summary.out
      touch $@


dds :       ${MAKE_SCR_DIR}/SLEEP_SOME_1   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_2   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_3   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_4   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_5   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_6   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_7   \
            ${SYN_DB_DIR}/RcvDDS.db
      touch $@


demod :     ${MAKE_SCR_DIR}/SLEEP_SOME_1   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_2   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_3   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_4   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_5   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_6   \
            ${MAKE_SCR_DIR}/SLEEP_SOME_7   \
            ${SYN_DB_DIR}/DeMod.db
      touch $@

RCV_CORE : ${SYN_DB_DIR}/Rcv_Core.db
      touch $@

XMT_CORE : ${SYN_DB_DIR}/Xmt_Core.db
      touch $@

#########################################################################

MAP_EFFORT      = medium
SYN_LIB_NAME    = OM73_lib
CLK_PERIOD2     = 500
CLK_PERIOD5     = 180
CLK_PERIOD20    = 40
CLK_PERIOD75    = 12
CLK_PERIOD90    = 10
CLK_PERIOD100   = 8
CLK_PERIOD150   = 5
SYN_UNIQUIFY    = 0
SYN_UNGROUP     = 0
```

```
ver             = 1.0
PARAMETERS      = none
Nm              = 4

${MAKE_SCR_DIR}/SLEEP_SOME_1 : ${MAKE_SCR_DIR}/bed_check
      sleep 135
      touch  $@

${MAKE_SCR_DIR}/SLEEP_SOME_2 : ${MAKE_SCR_DIR}/bed_check
      sleep 270
      touch  $@

${MAKE_SCR_DIR}/SLEEP_SOME_3 : ${MAKE_SCR_DIR}/bed_check
      sleep 405
      touch  $@

${MAKE_SCR_DIR}/SLEEP_SOME_4 : ${MAKE_SCR_DIR}/bed_check
      sleep 540
      touch  $@

${MAKE_SCR_DIR}/SLEEP_SOME_5 : ${MAKE_SCR_DIR}/bed_check
      sleep 675
      touch  $@

${MAKE_SCR_DIR}/SLEEP_SOME_6 : ${MAKE_SCR_DIR}/bed_check
      sleep 810
      touch  $@

${MAKE_SCR_DIR}/SLEEP_SOME_7 : ${MAKE_SCR_DIR}/bed_check
      sleep 945
      touch  $@

#########################################################################
#
#   User Input Buffer blocks

${SYN_DB_DIR}/AmiDec.db  :  ${SYN_LIB_DIR}/AMIDEC.syn
      @echo " "; date
      ${WAIT_FOR_DC}
      ${SYNTH_NODE} "${DC_SETUP};                 \
            setenv ENTITY_NAME  AmiDec;            \
            setenv CLK_NAME     DecClk;            \
            setenv CLK_PERIOD   ${CLK_PERIOD90};   \
            setenv RESET_NAME   Reset_n;           \
            setenv MAP_EFFORT   high;              \
            setenv SYN_LIB_NAME ${SYN_LIB_NAME};   \
            setenv SYN_UNIQUIFY ${SYN_UNIQUIFY};   \
            setenv SYN_UNGROUP  ${SYN_UNGROUP};    \
            setenv SYN_PARAMS   ${PARAMETERS};     \
            setenv CUSTOM_SCR   0;                 \
            setenv SYNOPSYS_V_PAD  vsc${VLSI_VERSION}p31; \
            setenv SYNOPSYS_V_PART vsc${VLSI_VERSION}83;  \
            ${NICE_IT} dc_shell -f ${SYN_mSCR_DIR}/syn_def_comp.scr \
                  >! ${SYN_DB_DIR}/AmiDec.out"
```

```
#  note this block does not use default clk_period
#  clock period for this block is 500ns
${SYN_DB_DIR}/ESCint.db   :  ${SYN_LIB_DIR}/ESCINT.syn  \
                             ${SYN_LIB_DIR}/ESCINT__RTL.syn
      @echo " "; date
      ${WAIT_FOR_DC}
      ${SYNTH_NODE} "${DC_SETUP};                  \
            setenv ENTITY_NAME  ESCint;             \
            setenv CLK_NAME     Clk;                \
            setenv CLK_PERIOD   ${CLK_PERIOD2};     \
            setenv RESET_NAME   0;                  \
            setenv MAP_EFFORT   ${MAP_EFFORT};      \
            setenv SYN_LIB_NAME ${SYN_LIB_NAME};    \
            setenv SYN_UNIQUIFY ${SYN_UNIQUIFY};    \
            setenv SYN_UNGROUP  ${SYN_UNGROUP};     \
            setenv SYN_PARAMS   ${PARAMETERS};      \
            setenv CUSTOM_SCR   0;                  \
            setenv SYNOPSYS_V_PAD  vsc${VLSI_VERSION}p31; \
            setenv SYNOPSYS_V_PART vsc${VLSI_VERSION}83;  \
            ${NICE_IT} dc_shell -f ${SYN_mSCR_DIR}/syn_def_comp.scr \
                  >! ${SYN_DB_DIR}/ESCint.out"

${SYN_DB_DIR}/InputBuf_uPIF.db  :  ${SYN_LIB_DIR}/INPUTBUF_UPIF.syn
      @echo " "; date
      ${WAIT_FOR_DC}
      ${SYNTH_NODE} "${DC_SETUP};                 \
            setenv ENTITY_NAME  InputBuf_uPIF;     \
            setenv CLK_NAME     Clk;               \
            setenv CLK_PERIOD   ${CLK_PERIOD90};   \
            setenv RESET_NAME   SReset_N;          \
            setenv MAP_EFFORT   ${MAP_EFFORT};     \
            setenv SYN_LIB_NAME ${SYN_LIB_NAME};   \
            setenv SYN_UNIQUIFY ${SYN_UNIQUIFY};   \
            setenv SYN_UNGROUP  ${SYN_UNGROUP};    \
            setenv SYN_PARAMS   ${PARAMETERS};     \
            setenv CUSTOM_SCR   0;                 \
            setenv SYNOPSYS_V_PAD  vsc${VLSI_VERSION}p31; \
            setenv SYNOPSYS_V_PART vsc${VLSI_VERSION}83;  \
            ${NICE_IT} dc_shell -f ${SYN_mSCR_DIR}/syn_def_comp.scr \
                  >! ${SYN_DB_DIR}/InputBuf_uPIF.out"

#clock period for this block is 200ns
${SYN_DB_DIR}/dmux_e.db   :  ${SYN_LIB_DIR}/DMUX_E.syn \
                             ${SYN_DB_DIR}/ais_e.db    \
                             $(SYN_CUST_DIR)/dmux_e.scr
      @echo " "; date
      ${WAIT_FOR_DC}
      ${SYNTH_NODE} "${DC_SETUP};                  \
            setenv ENTITY_NAME  dmux_e;             \
            setenv CLK_NAME     T1_E1_CP;           \
            setenv CLK_PERIOD   ${CLK_PERIOD5};     \
            setenv RESET_NAME   RESET;              \
            setenv MAP_EFFORT   ${MAP_EFFORT};      \
```

```
            setenv SYN_LIB_NAME ${SYN_LIB_NAME};    \
            setenv SYN_UNIQUIFY ${SYN_UNIQUIFY};    \
            setenv SYN_UNGROUP  ${SYN_UNGROUP};     \
            setenv SYN_PARAMS   ${PARAMETERS};      \
            setenv CUSTOM_SCR   1;                  \
            setenv SYNOPSYS_V_PAD  vsc${VLSI_VERSION}p31; \
            setenv SYNOPSYS_V_PART vsc${VLSI_VERSION}83;  \
            ${NICE_IT} dc_shell -f ${SYN_mSCR_DIR}/syn_def_comp.scr \
                  >! ${SYN_DB_DIR}/dmux_e.out"
######################################################################
```

FIGURE 4 - Part of the Makefile

The makefile provided a mechanism for setting up parameters used in the default synthesis script. In this manner, a single base script was used for synthesizing the majority of the project. When the makefile was called to build the design, it first built a number of SLEEP targets. The purpose of these targets was to offset the selection of workstations. When the makefile began, the first eight non-dependent targets were built. If all eight Synopsys licenses were immediately available at the beginning of the makefile run and if the eight makefile targets were all targets to be synthesized, then all eight would start at the same time and check the network for the best workstation at the same time. All eight would then start on the same workstation. By forcing the first seven rules of the makefile to be SLEEP processes, the parallel starts of the synthesis compiles were offset. The offset time was long enough to allow previously started jobs to load the workstation they were running on. The scripts to monitor the workstations are home grown and work well most of the time. There are a number of free and commercial programs available that perform similar tasks such as Load Balancer, Load Leveler, DQS and LSF. The best way to get information about these programs is to look at the following web site:

Batch Queuing Systems:
http://www.cmpharm.ucsf.edu/~srp/batch/systems.html

This web site has a link to a paper by NASA that compares the features of many of the queuing packages available when that paper was written. That paper is a great starting point to learn about the tools and features available. The paper is titled: "A Comparison of Queuing, Cluster and Distributed Computing Systems."

The makefile is executed by running a simple executable file:

```
#! /bin/csh -f

date

touch $PROJECT/make_control/bed_check

gmake -j 8 -e VLSI_VERSION=7 -f  /user/om73/Makefile.synth > \
                  /user/om73/make_control/mksyn783.out
```

```
date

echo /user/om73/make_control/mksyn783.out
```

FIGURE 5 - UNIX Script to Execute Parallel Make

This file was usually executed using the UNIX ‘at’ command. Note that before running the makefile, the file called bed_check was touched. This caused all the sleep targets in the makefile to be out of date and forced them to be built when the makefile was executed.

Link and Characterize

The magnitude of this project was such that it had to be broken down into hundreds of small entities. The grouping of these entities for synthesis compiling was at the 5000 to 10,000 gate count, or at the 5-8 hour run time. Because of the parallel nature of the processing, it was acceptable to allow one job to run for an extended period of time, since it did not block the remaining parts of the ASIC from being compiled. As the compilation of the 5000 to 10,000 gate blocks completed, an additional default script was developed to link these blocks together. This script is very similar to the compile script except with the compile command replaced by the link command. Initially, no efforts were made as part of the linking to buffer the interconnection of the compiled sub-blocks. This buffering took place using a special characterize-compile script that traversed the hierarchy only as directed and buffered the designs. Hence, the reason for holding off the buffering of the sub-block interconnections was to wait until the top level was in place and then execute this characterize-compile script.

The characterize-compile script was set up to characterize one sub-block at a time within the current design. Once that sub-block was characterized, then that sub-block was entered and dont_touch attributes were placed on all the cells within that design. The design was then compiled and buffers were added as needed to meet constraints of the design. The current_design was then set back to the top level and the process repeated on the next block. This continued until all the blocks (at the level where characterize-compile was executed) have been characterized and compiled. The characterize-compile script is written generically to extract the cell names and references for the blocks it was executing on. Thus, this one script to be used anywhere in the project hierarchy that needed to be characterized, without any additional modification to the script. The only drawback to this approach was that there was no control over the order in which the sub-blocks were characterized. The designs and script could probably have had attributes added to them to control the order of characterization, but the script in its current form provided sufficient results.

This characterize-compile script has one additional feature that must be discussed in detail. When this script was run at a top level of the design, there were some sub-blocks that were characterized and compiled that were quite large and had many sub-blocks at their level.

Therefore, in order to buffer this lower hierarchical sub-level and its sub-blocks, the characterize-compile script was recursively called. The characterize-compile script contains a variable called “sub_design_list”. The script compared the sub-block being characterized with those listed in the sub_design_list. If there was a match, then after characterization and the sub-block being entered, the script was again called and executed on this sub-block. When the sub-block characterize-compile script completed, the original running script then continued. This approach provided control over traversing branches and depth of the hierarchical design tree.

```
/*
    this script is to characterize/compile the current design and then
    recursively call itself on sub-levels if that sub-level is on the
    sub_level_list


    In order to call this script the following commands must be executed
    in the script which calls this file...


    temp = ""
    current_design
    temp = dc_shell_status
    stack_file =  execute (-s "sh echo temp | sed 's![{}]!!g'") + "_stack"
    sh touch stack_file

    include /user/om73/make_control/syn_default/syn_def_char_comp.scr

    sh rm stack_file


    This assures a name is set up for the stack used as part of the recursion.

*/

sub_design_list = {top_fir2      \
                   dec_fir       \
                   down_filt     \
                   acs           \
                   VitNoTCM      \
                   vit_tcm       \
                   vcbtcm_e      \
                  }

sh date

top = current_design

report_timing -path full -delay max -max_paths 5 -nworst 1 \
              -to all_registers(-data_pins) + all_outputs()
report_area

filter find(cell "*") "@is_hierarchical == true"
hier_cells = dc_shell_status
```

```
foreach(inst, hier_cells) {
  sh date
  list inst
  current_design = top
  /*
  report_timing -path full -delay max -max_paths 5 -nworst 1 \
                -to all_registers(-data_pins) + all_outputs()
  */
  characterize -constraints inst
  get_attribute inst ref_name -quiet
  current_design dc_shell_status
    current_design
    sub_design_name = dc_shell_status
    foreach(item, sub_design_list) {
      list item
      if (item == sub_design_name) {
        /* push current value of top onto the stack */
        echo top >> stack_file

          include /user/om73/make_control/syn_default/syn_def_char_comp.scr

        /* pop current level of top from the stack  */
        execute -s "sh tail -n 1 stack_file"
        top = dc_shell_status
        sh mv stack_file moo_temp
        sh sed '$d' moo_temp > stack_file
        sh rm moo_temp
      }
    }
    report_area
    /*
    report_timing -path full -delay max -max_paths 5 -nworst 1 \
                  -to all_registers(-data_pins) + all_outputs()
    */
    set_dont_touch find(cell, "*")
    compile -only_design_rule
    /*
    report_area
    report_timing -path full -delay max -max_paths 5 -nworst 1 \
                  -to all_registers(-data_pins) + all_outputs()
    */
    set_dont_touch find(cell, "*") false
}

current_design = top
report_timing -path full -delay max -max_paths 5 -nworst 1 \
              -to all_registers(-data_pins) + all_outputs()
report_area
```

FIGURE 6 - Recursive Characterize-Compile Script

In order to use this script, the setup commands described in the header must be executed prior to calling the script. These commands set up a stack used by the script when traversing the hierarchy. This script contains a number of report_timing commands that are commented out. These timing reports were used to monitor the progress of the characterize-compile script. The results of using recursion to characterize-compile this project were mixed. The recursive process turned out to be a very slow process since each sub-block had to be dealt with uniquely. However the results were quite good. To speed the process up, the characterize-compile script was applied directly to lower levels in the hierarchy rather than traverse the hierarchy to those levels. This approach gave acceptable timing results (though not as good as the recursive approach) and executed much faster. The ability to run the lower level characterize-compile scripts in parallel was a large factor to speeding up the overall run time.

Miscellaneous Scripts

There are three final scripts that were used on the project. The first was used to write out the design in VHDL structural format. The basis for this script came from SOLV-IT but was modified to accommodate the parallel nature of the design flow.

write out VHDL....

```
/* write out the VHDL for the design... Since Synopsys does not have a switch
for VHDL like it does for Verilog to write out a hierarchical design in the
correct order to a single file, this script performs the task.  It writes out
all the designs in memory and keeps track of the order.  After all the designs
are written out, the files are concatenated together in to a  signals file
using UNIX utilities and the file order determined when writing out the files.
Note that this script seems to choke on black boxes such as memories or
entities with no arch.

*/

/*  currently don't want to remove design
remove_design -all
*/

/* vhdl_file is a variable that contains the path to the vhdl file containing
the full hierarchy for the current design.  This variable is set in the
include script above... vhdl_bottoms_up.scr
*/

/* dont need to read the design back since it was not removed
read -f vhdl vhdl_file
*/

/*****************************************************************************/

temp = ""
current_design
temp = dc_shell_status
vhdl_file = "/user/om73/synopsys_v3.4b/gate_level_vhdl_from_syn/" + \
```

```
             execute (-s "sh echo temp | sed 's![{}]!!g'") + ".vhdl"
sh touch vhdl_file

write -hier -f vhdl
sh "mv *.vhd ~om73/synopsys_v3.4b/gate_level_vhdl_from_syn/."

ordered_des_list = {}
all_designs = find(design,"*")

while(all_designs){
    foreach(present_design, all_designs){
            current_design present_design
            filter find(cell,"*") "@is_hierarchical==true"
            sub_list = dc_shell_status
            foreach (subdesign,sub_list) {
                    reference = get_attribute(subdesign,ref_name)
                    foreach (item,ordered_des_list) {
                            if (item == reference) {
                                    sub_list = sub_list - subdesign
                            }
                    }
            }
            if (sub_list == {}) {
                    ordered_des_list = ordered_des_list + present_design
                    all_designs = all_designs - present_design
            }
    }
}

sh rm vhdl_file
sh touch vhdl_file

foreach(each_entity,ordered_des_list){
    sh cat "/user/om73/synopsys_v3.4b/gate_level_vhdl_from_syn/" + \
            each_entity + ".vhd >> " vhdl_file
}
```

FIGURE 7 - Write VHDL Correct Order Script

The main change to the original script was to make the vhdl_file variable unique so that this script could be executed by concurrently running Synopsys jobs.

The final two scripts presented here are scripts to buffer the scan_enable line inserted by DFT methodology. With the new tools from Synopsys these scripts will not be needed, but until everyone gets this new software, these scripts could be useful. They take two different approaches to buffering the scan enable. One performs a hierarchy compile and the other compiles each block of the hierarchy uniquely.

```
sh date
```

```
/* This script will buffer each design in memory except for those designs
listed in the dont_buffer_list
*/

dont_buffer_level = {DeMod         \
                     demod_control \
                     Decode_Demux  \
                     vcbtcm_e      \
                    }

set_max_transition 3 find(design,"*")
set_driving_cell -cell dfntsq1 -pin q all_inputs()

foreach(block, find(design)) {
    list block
    do_buffer = 1
    foreach(item, dont_buffer_level) {
        list item
        if (item == block) {
            do_buffer = 0
        }
    }
    list do_buffer
    if (do_buffer == 1) {
        current_design block
        find(port, scan_en)
        scan_en_exists = dc_shell_status
        list scan_en_exists
        foreach(scan_item, scan_en_exists) {
            if (scan_item == scan_en) {

                all_nets         = {}
                scan_enable_nets = {}
                dont_touch_nets  = {}

                set_dont_touch find(cell, "*")

                find(net, "*")
                all_nets = dc_shell_status

                find(net, "*sc*n_en")
                scan_enable_nets = dc_shell_status

                dont_touch_nets = all_nets - san_enable_nets

                set_dont_touch dont_touch_nets
                set_dont_touch scan_enable_nets false

                set_driving_cell -cell dfntsq1 -pin q all_inputs()
                set_max_transition 3 block

                compile -only_design_rule

                set_dont_touch find(cell, "*") false
```

```
                set_dont_touch find(net,  "*") false
            }
        }
    }
}

sh date
```

FIGURE 8 - Block Level Scan_Enable Buffering

```
/* This script will hierarchically buffer the scan_enable.  The designs listed
the design_list are the top level hierarchy starting point if those designs
are in memory.
*/
sh date
design_list = {demod_control dec_fir_scan down_filt}
foreach(design, design_list ) {
    current_design design

    all_nets         = {}
    scan_enable_nets = {}
    dont_touch_nets  = {}

    set_dont_touch find(cell, "*", -hier)

    find(net, "*", -hier)
    all_nets = dc_shell_status

    find(net, "*sc*n_en", -hier)
    scan_enable_nets = dc_shell_status
    dont_touch_nets = all_nets - san_enable_nets

    set_dont_touch dont_touch_nets
    set_dont_touch scan_enable_nets false

    set_driving_cell -cell dfntsq1 -pin q all_inputs()

    compile -only_design_rule

    set_dont_touch find(cell, "*", -hier)
    set_dont_touch find(net,  "*", -hier)
}
sh date
```

FIGURE 8 - Block Level Scan_Enable Buffering

Summary

The magnitude of the synthesis processing for this large project was greatly reduced and became manageable because of the use of parallel make and parameterizable scripts. The constraints placed on the coding of the design greatly helped the automation of the process. There were

many places where this process could be improved, such as replacing some of the home grown scripts with real (supported) tools that would perform the tasks much cleaner. Overall this automation process brought success to the project. The 260 synthesis runs took around 40 hours to complete while utilizing eight Synopsys compiler licenses.