DARClab Newcomer's Manual

Version	Date	Prepared by	Comments
1.0	18/8/2014	BCS	Initial document
1.1	5/9/2016	BCS	Changes for UTD

Welcome to PolyU's DARClab (Design Automation and Reconfigurable Computing Laboratory). This document describes the most important software that you will need in order to efficiently progress with your project and how to set it up. You can also visit your department's YouTube channel for videos regarding how to use some of the EDA tools and how to set up the machines at www.youtube.com/user/DARClabify.

1. Connect your local Windows PC to the DARClab's Linux machines using Cygwin

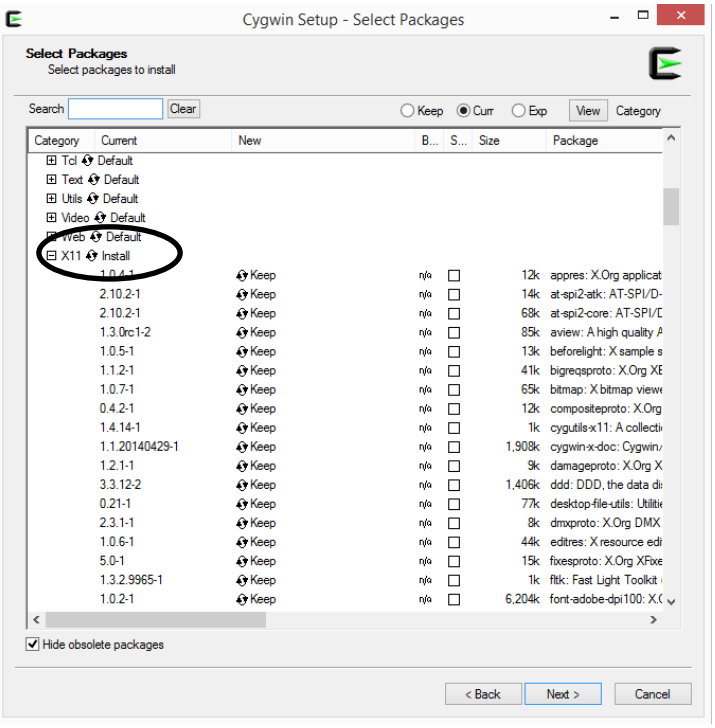
The DARClab has currently 3 Linux servers on which the EDA tools run: (1) tools.eie.polyu.edu.hk, (2) legolas.eie.polyu.edu.hk and (3) gandalf.eie.polyu.edu.hk.

The tools machine is used as a gateway and is the only one through which you will be able to connect to our Linux system.

> **Note:** Because tools is an old machine do not used it to run long simulations/synthesis. The legolas and gandalf machines are the fastest and should be normally used for long jobs.

To log into the system you will need to install Cygwin from www.cygwin.com. Make sure you install the correct version based on your OS, setup-x86.exe for 32-bit or setup-x86_64.exe for 64-bits.

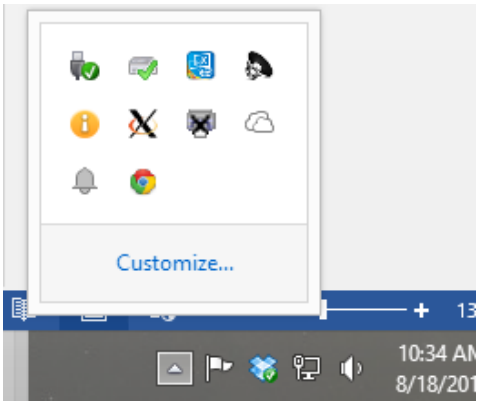
When you install Cygwin, make sure to select the full X11 packages at the end of the package list (click on the X11 package family until "install" appears)



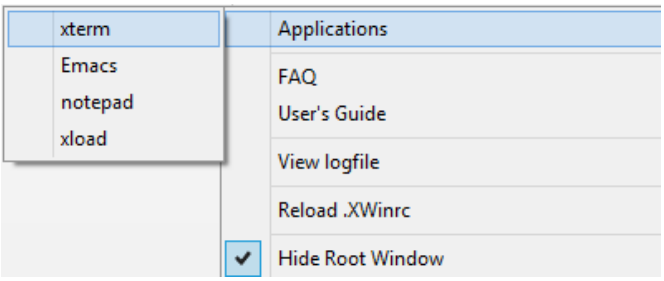
The installation process might take some time depending on the server selected and the internet traffic.

By default Cygwin will be installed in c:\cgwin. In the bin directory you will find the xlaunch.exe and xwin.exe binaries. The first allows you to configure the connection and the latter starts the Windows server. You can also create a shortcut on your desktop, e.g. “C:\cygwin64\bin\run.exe -p /usr/X11R6/bin XWin –multiwindow”

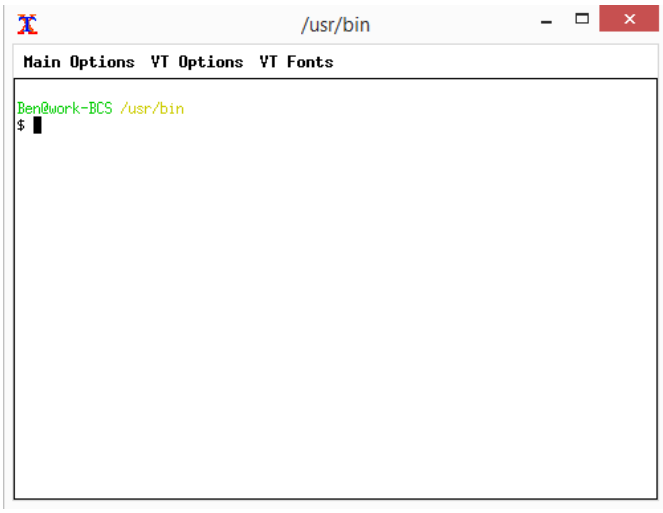
Once Cygwin is installed, you can connect to the DARClab’s Linux machines if the system administrator has created an account for you. Click on XWin. It will launch the XWin server as shown below:



Right-click on the XWin icon and launch a terminal (xterm)



This will open a terminal window as shown below:



To connect to the Linux machines you need to connect to the tools server once you have a valid account. In order to allow applications with a graphical user interface to be displayed on your local host you need to type the following BEFORE you try to connect.

```
$xhost +
```

Then log using ssh as follows:

```
$ssh username@tools.eie.polyu.edu.hk
```

Enter your password and you will be connected to your home directory at the tools machine.

You can FTP files between the Windows and Linux environment by downloading one of the many free FTP clients available online e.g. WinSCP (http://winscp.net/eng/index.php)

2. Setting up the Linux Environment

One of the most important files in Linux is the .bashrc file in your home directory. It is loaded every time you log in and contains the environment variables and paths to all the tools. Every time this file is modified it needs to be reloaded (sourced):

```
username@tools:~>source .bashrc
```

If it is the very first time you log into the system you will need to add the following paths to your .bashrc file in EACH of the 3 machines (tools, legolas and gandalf) separately. You can use Linux's vi text editor or xemacs

```
username@tools:~>xemacs ~/.bashrc &
```

```
# Automatically set DISPLAY variable
if [ ! $DISPLAY ] ; then
   if [ "$SSH_CLIENT" ] ; then
      export DISPLAY=`echo $SSH_CLIENT|cut -f1 -d\ `:0.0
   fi
fi

# Set path for Altera
export PATH=$PATH:/eda/bin/altera/12.0/quartus/bin
export PATH=$PATH:/eda/bin/altera/12.0/nios2eds/bin

export LM_LICENSE_FILE=$LM_LICENSE_FILE:18000@eda.eie.polyu.edu.hk

#Set path for Xilinx
export PATH=$PATH:/eda/bin/xilinx/Vivado/2012.2/bin #vivado
export PATH=$PATH:/eda/bin/xilinx/Vivado_HLS/2012.2/bin #vivado_hls
export PATH=$PATH:/eda/bin/xilinx/14.3/ISE_DS/ISE/bin/lin  #ise
export PATH=$PATH:/eda/bin/xilinx/14.3/ISE_DS/common/bin/lin  #ise utilities - license mananger xlcm

export LM_LICENSE_FILE=$LM_LICENSE_FILE:2100@eda.eie.polyu.edu.hk

#Set path for Mentor's Modelsim
export PATH=$PATH:/eda/bin/altera/12.0/modelsim_ase/bin

#set SystemC path
#export PATH=$PATH:/eda/bin/systemc/systemc-2.3.0
#export SYSTEMC_HOME=/eda/bin/systemc/systemc-2.3.0
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SYSTEMC_HOME/lib_linux

export PATH=$PATH:/eda/bin/cwb/cyber/osci
export SYSTEMC_HOME=/eda/bin/cwb/cyber/osci
export TLM_HOME=$SYSTEMC_HOME/TLM-2009-07-15
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SYSTEMC_HOME/lib-linux

#Version CWB 5.4
export CYBER_PATH=/eda/bin/cwb/cyber_540/LINUX

export CYBER_SYSTEMC_HOME=${CYBER_PATH}/osci
export CYBER_LIB=${CYBER_PATH}/lib
export LD_LIBRARY_PATH=${CYBER_PATH}/lib:${LD_LIBRARY_PATH}

export CYLMD_LICENSE_FILE=27000@eda.eie.polyu.edu.hk
export LM_LICENSE_FILE=$LM_LICENSE_FILE:27000@argus.eie.polyu.edu.hk

export PATH=$PATH:${CYBER_PATH}/bin
```

```
#Graphviz
export PATH=$PATH:/eda/bin/xilinx/Vivado_HLS/2012.3/Linux_x86/tools/graphviz/bin

#Aldec Riviera-Pro
export PATH=$PATH:/eda/bin/aldec/riviera-pro-2012.10/bin
export ALDEC_LICENSE_FILE=27009@eda.eie.polyu.edu.hk
```

Save the changes and source the new .bashrc file

Log in to legolas.eie.polyu.edu.hk and gandalf.eie.polyu.edu.hk and modify the .bashrc file as done for the tools machine. To log into the other machines:

username@tools:~>ssh -Y <username>@legolas.eie.polyu.edu.hk

and

username@tools:~>ssh -Y <username>@gandalf.eie.polyu.edu.hk

Note: Do not forget the option "-X" when calling ssh. This option re-directs the graphical display to your local machine.

You need to update the .bashrc file with

```
# Automatically set DISPLAY variable
if [ ! $DISPLAY ] ; then
   if [ "$SSH_CLIENT" ] ; then
      export DISPLAY=`echo $SSH_CLIENT|cut -f1 -d\ `:0.0
   fi
fi
```

to allow GUI forwarding

The legolas.eie.polyu.edu machine has been setup as a fileserver. That means that the files stored on this machine will be automatically mounted on the tools and Gandalf machine when logged in. Therefore when logging into tools or gandalf you will encounter two home folders:

/home/<username>
/home_mnt/<username>

The best way to work more efficient is to create a folder at the /home directory and create a symbolic link between this folder and the /home_mnt folder. This will link the folder's contents to the legolas machine file structure and you will be able to access all the legolas' files from any machine. E.g.

username@legolas:~>mkdir common

Log in to tools and gandalf:

username@tools:~>ln –s /home_mnt/<username>/common common

The symbolic link can anytime be deleted without deleted the files it is pointing to. Also a single folder within legolas can be shared this way.

You can also move files across the machines using the scp command. E.g. for moving a complete folder form the tools to Gandalf machine.

username@tools:~> scp –r /folder <username>@gandalf.eie.polyu.edu.hk:/home/<username>/<dest>

3. Software development in Linux

All of the design automation software done at the laboratory will be developed on the Linux machines. The following YouTube video contains a similar context as this section https://www.youtube.com/watch?v=B6o4oX_ZrjE

Most of the SW will be written in ANSI-C/C++. The main tools used will be:

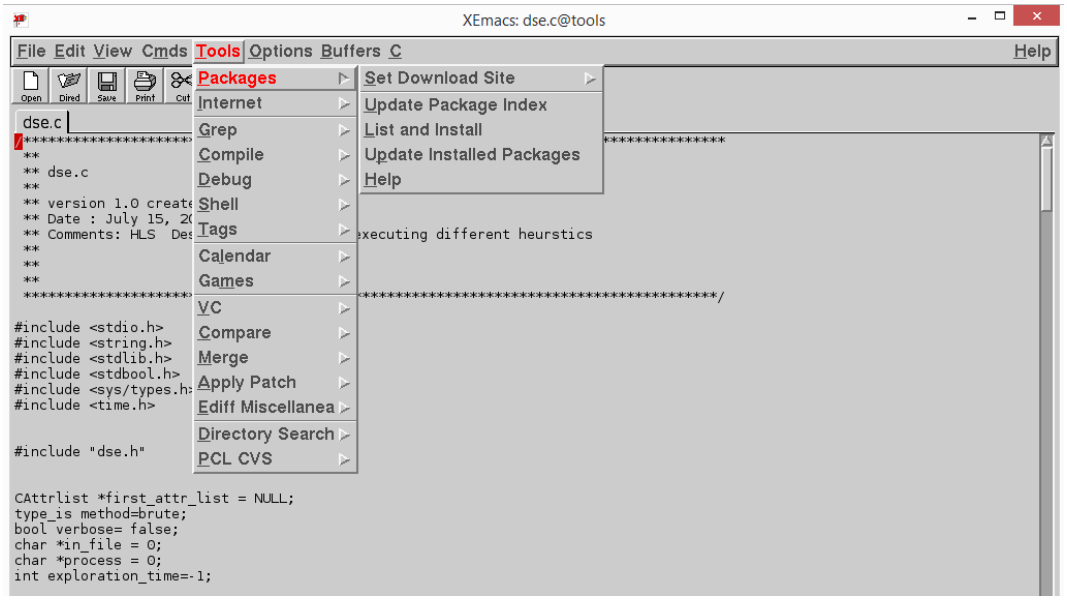
- Text editor to write the Program (typically xemacs)
- Compilers gcc for ANSI-C compilation and g++ for C++ (free in Linux)
- make. Linux make utility, which reads a Makefile and calls gcc
- gdb for debugging (free in Linux)
- ddd (graphical user interface of gdb)

Xemacs/emacs:

Xemacs is a highly configurable text editor. Launch it by typing:

```
username@tools:~>xemacs filename.c &
```

Xemacs might be difficult to use at the very beginning because the shortcuts are different from the typical text editors, but once you master it, it will become extremely powerful. It can be configured by downloading different packages to e.g. highlight source code:



This will create a configuration file in your home directory:

```
username@tools:~>ls ./xemacs/init.el
```

This file can be manually configured or overwritten. Set xemacs up in a way that you fill comfortable working with. You can e.g. google other init.el files and overwrite yours.

Makefile:

A Makefile contains the instructions for Linux's 'make' utility to compile a given program. Whenever you have finished writing a program, create a Makefile as follows

Note: Makefiles can be very complicated and have many options. The syntax is also very strict e.g. A tab is not the same as a space

```
# Source, Executable, Includes, Library Defines

TARGET=   dse.exe

INCL   = \
                  dse.h
SRCS    =  \
                  dse.c  \
                  ant_colony.c

OBJS    = $(SRCS:.c=.o)

# Compiler, Linker Defines
CC = /usr/bin/gcc
LINKER = $(CC)
CFLAGS= -ansi -pedantic -Wall -O3

debug : CFLAGS += -g -DDEBUG
RM      = /bin/rm -f

# Link all Object Files with external Libraries into Binaries
$(TARGET): $(OBJS)
                  $(LINKER) -o "$@"  $(OBJS)

# Create a gdb/dbx Capable Executable with DEBUG flags turned on
debug: $(OBJS)
                  $(LINKER) -o $(TARGET)  $(OBJS)

dse.o: dse.c
                  $(CC) $(CFLAGS)  -c $< -o $@

ant_colony.o: ant_colony.c
                  $(CC) $(CFLAGS) -c $< -o $@

# Clean Up Objects, Exectuables, Dumps out of source directory
clean:
                  $(RM)  $(OBJS) $(TARGET)
```

The Makefile is useful especially for larger projects and when different versions of the same program need to be created. E.g. a debug version and a release version. In this case when wanting to debug the program the following would be typed:

```
username@tools:~>make debug
```

This compiles the source code including the “debug” options specified. In this case “-g” and “–DDebug”.

-g: debug command for gcc. Generates the debug version of the program which can be debugged using gdb/ddd

-DDebug: gcc compiles the source code enabling any source code within the DEBUG pre-copmiler directive. E.g.

```
#ifdef DDEBUG
        printf(“\nThis part of the code will be displayed in the debug version\n”);
#else
       printf(“\nNow we are in the release version\n”);
#endif
```

To delete all object files type:

```
username@tools:~>make clean
```

Note: If the file is not called Makefile you need to type: $make –f <name_of_file> debug

gdb/ddd

gdb is the most widely used software debugger in Linux. It is command based and therefore not very easy to use. For this purpose ddd was developed as a GUI for gdb. To debug a software application using ddd you need to compile first the program with the debug option “-g” enabled and type:

```
username@tools:~>ddd dse.exe &
```

This will open the debugger where you can set breakpoints, display variables values, etc….

4. EDA Tools

The main EDA tools installed on the Linux system are:

Vendor	Tool name	Launch tool	Description
NEC	CyberWorkBench	%cwb &	High-Level Synthesis tool
Xilinx	ISE	%ise &	FPGA's Logic synthesis for Xilinx FPGAs
Altera	Quartus	%quartus &	Logic synthesis Altera's FPGA synthesis
Aldec	Riviera-PRO	%riviera &	RTL simulator

The GUI of these tools should be displayed on your remote machines if you have specified:

- %xhost + once before logging to the Linux system
- specifying "ssh –X" each time you connect from one machine to another
- Setting automatically the DISPLAY variable in the .bashrc

CyberWorkBench (CWB):

CWB is a large tool comprised of many binaries all integrated other the IDE. When creating your own automation programs, you will need to call these tools within your C/C++ program. The most important tools are:

Tool	Input	Output	Example	Description
scpars	SystemC file	.IFF file	%scpars foo.cpp	SystemC parser
bdlpars	ANSI-C/BDL fil	.IFF fil	%bdlpars foo.c	ANSI-C/BDL parser
bdltran	.IFF	_C.IFF/_E.IFF	%bdltran –c1000 –s foo.IFF –Zflib_fcnt_out	Main synthesis engine. Needs to be called 2 times (see below for explanation)
veriloggen	_E.IFF	_E.v	%veriloggen foo_E.IFF	Bdltran Verilog generation backend
vhdlgen	_E.IFF	_E.vhd	%vhdlgen foo_E.IFF	Bdltran VHDL generation backend
cmscgen	_C.IFF	.cpp	%cmscgen foo_C.IFF	Generates cycle-accurate SystemC model for verification
tbgen	_E.IFF	.v/.vhd	%tbgen foo_E.IFF	Generates RTL testbench

LSscrgen	_E.IFF	.tcl	%LSscrgen foo_E.IFF	Generates synthesis script for logic synthesizer (DC, ISE, Quartus, etc..)

More information about each of these tools can be found in the user manuals or by typing the tool's name –h. E.g. %scpars -h

Bdltran is the main synthesis engine and needs to be called twice. The first time if generates the Functional Units (FUs) resource constraint file and the second it uses this file to synthesizes the description.

To manually synthesize a design from the command file:

%bdlpars foo.c
>> Generates foo.IFF file

%bdltran –c1000 –s –lfl asic_45.FLIB –lb asic_45.BLIB foo.IFF –Zflib_fcnt_out –Zmlib_mcnt_out
>> Generates foo-auto. FCNT (resource constraint file) , foo-auto.FLIB (if needed, normally empty), foo-auto. MLIB and foo-auto.MCNT (memory constraint files if needed)

%bdltran –c100 –s –lfl asic_45.FLIB –lfc foo-auto.FCNT –lb asic_45.BLIB –lml foo-auto.MLIB –lmc foo-auto.MNCT –foo.IFF
>> Synthesizes the description and generats foo_C.IFF (result after scheduling) and foo_E.IFF (result after binding).

%veriloggen foo_E.IFF
>> Generates foo_E.v

5. Writing your first ANSI-C Program for EDA

Often it is required to call third party tools from within an ANSI-C program. In this case use the "system" call. E.g.

```
int main(){
                char buffer[];
                strcpy(buffer," bdlpars foo.c");
                system(buffer);

}
```

6. Linux Scripts

Writing scripts can help you being more productive. E.g. when having to run many simulations, a small shell script can be setup to call your program with different input parameters:

Shell script example:

```
%touch experiments.csh
%vi experiments.csh
```

```
#!/bin/csh
# experiment 1
../heuristic/dse.exe –parameter1 X –parameter2 Y
mv results.txt results_experiment1.txt

#experiment2
../heuristic/dse.exe –parameter1 N –parameter2 M
mv results.txt results_experiment2.txt
```

Execute by:

```
%chmod  +x experiments.csh
%./experiments.csh
```

Perl scripts are also very useful to parse text files and extract information from them. An example could be:

```
#! /usr/bin/perl
##############################################################################################
# PolyU PROPRIETARY
#
# File Name             extract_runtime.pl
# Function              extracts the runtime of exploration runs from log file
##
```

```
# Author                  Benjamin Carrion Schafer
# Date                  July 18, 2013
# Change history
#
#  Usage example: perl extract_runtime.pl -log <filename> -logout <name>
#
##########################################################################################

#use strict;
use warnings;
use diagnostics;
use POSIX;
use File::Basename;

sub main () {

print "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~";
print "\n Welcome to PolyU Runtime Extractor CWB V1.0";
print "\n";
print "\n Written by Benjamin Carrion Schafer";
print "\n Copyright PolyU ";
print "\n Version 1.0, July 18, 2013";
print "\n";
print "\n -h for help";
print "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n";

my $argc = @ARGV;
my $temp = "";
$logfile = "";
$logfile_out = "";

#######  READ  input arguments
for ($i=0 ;$i< $argc; $i++)
  {

    $temp = $ARGV[$i];

    if(!$temp){
      print "\n\t ERROR: Missing arguments needed. Check help menu \n";
      &help();
    }

    # Version print
    if($temp eq "-v") {
      exit -1; # version printed in header
    }
    # -h Print the help menu
   if($temp eq "-h"){
     &help();
    }

    # -log Specifies the Input log file
   if($temp eq "-log"){

      if ($i+1 <$argc){
                        $logfile = $ARGV[$i+1];
      }
      else{
                        print "\n\t ERROR: Missing $temp argument. Check help menu \n";
                        &help();
      }
    }

    # -logout Specifies the output log file
   if($temp eq "-logout"){

      if ($i+1 <$argc){
```

```
                $logfile_out = $ARGV[$i+1];
    }
    else{
                print "\n\t ERROR: Missing $temp argument. Check help menu \n";
                &help();
    }
   }
  }


   # Check that all needed  inputs where specified
   if($logfile eq ""){
                print "\n\t ERROR: Missing argument <log file>  Arguments specified:\n";
     print "\n\nCheck help menu: \n";
                &help();

    }


  ###############
  ##  Extract runtime of explored designs
  ##
  ## log file format:
  ## RUN 4 / 1
  ##DESIGN ./history/001a ave8 2013/07/18 15:27:20 ATTR 0020 OPTS 1        L:1 M:3 S:4 AREA    4482 LATENCY    2
CP_DELAY    1.62 CYCLES_SIM      0     OK
  ##DESIGN ./history/024a ave8 2013/07/18 15:29:05 ATTR -0023449 OPTS 1        L:1 M:3 S:1 AREA    6310 LATENCY
12 CP_DELAY    3.17 CYCLES_SIM      0  NOTPARETO
  ##DESIGN ./history/026a ave8 2013/07/18 15:29:20 ATTR -0022653 OPTS 1        L:1 M:3 S:1 AREA    6741 LATENCY
10 CP_DELAY    3.17 CYCLES_SIM      0  NOTPARETO
  ##DESIGN ./history/027a ave8 2013/07/18 15:29:28 ATTR -0023262 OPTS 1        L:1 M:3 S:2 AREA    6701 LATENCY
10 CP_DELAY    3.17 CYCLES_SIM      0  NOTPARETO

   &extract_designs();

}

&main();

 ############################
 ## Extract the designs' synthesis runtime
 ##

sub extract_designs(){

   if($logfile_out eq ""){
     $report_file = "runtime_report_log.xls";
   }
   else{
     $report_file = "runtime_" . $logfile_out .".xls";
   }


   # Generate report file to store a summary of the system
   open(REPORT_FILE_XLS, ">$report_file") || die("ERROR: Could not generate report file $report_file");

   print REPORT_FILE_XLS "\n\nRuntime Analysis";

   ############################
   ## Read log file
   ##
   ##
   open(REPORT, $logfile) || die("ERROR: Could not open CWB report file $logfile");
   @log_file = <REPORT>;
  close(REPORT);
```

```
  $flag_bdltranerror2 = 0, $flag_bdltranerror1=0;
  $start=0;
  $end =0;

  $design;
  $time_start;
  $time_finish;

 foreach $log_line  (@log_file){

   chomp($log_line);

   if($log_line =~ /DESIGN/){

     @line = split(/\s+/, $log_line);
     @time_log = split(/:/, $line[4]);

     if($start == 0){
                        $start = ($time_log[0]*3600)+ ($time_log[1]*60) + $time_log[2];

#                       if($log_line =~ /BDLTRANERROR/){
#                         $flag_bdltranerror1 = 1;
#                       }

                        if($line[13] eq 0){
                          $flag_bdltranerror1 = 1;
                        }
                        $design = $line[1];
                        $time_start = $line[4];
     }
     else{
                        $end = ($time_log[0]*3600) +($time_log[1]*60) +  $time_log[2];
                        $runtime = $end - $start;
                        $time_end = $line[4];

                        if($line[13] eq 0){
                          $flag_bdltranerror2 = 1;
                        }
                        else{
                          $flag_bdltranerror2 = 0;
                        }

                        if($runtime > 0 and $flag_bdltranerror1 == 0 ){
                          print REPORT_FILE_XLS "\n$design\t$line[2]\t$line[4]\t$time_start\t$time_end";
                          print REPORT_FILE_XLS "\t$runtime";
                          print "\n$design $line[2] $line[4]  $end - $start = $runtime  $time_start  $time_end";

                        }
                        $start = $end;
                        $design = $line[1];
                        $flag_bdltranerror1=$flag_bdltranerror2;
                        $flag_bdltranerror2 = 0;
                        $time_start = $time_end;
                        $time_end = 0;
     }
    }
  }


   print "\n\n";

   close(REPORT_FILE_XLS);


exit -1;

 }
```

```
#########################
##
## Print help
##
##
sub help()
{
  print "\nHELP MENU ";
  print "\nlog <name>\t: Name of log file to be analuzed";
  print "\nlogout <name>\t: Name of output xls report file";

  printf "\n\nCommand format is:\n\tperl extract_runtime -log run_attr.log -logout ave8 \n\n";
  exit -1;

}
```

[END]